공학석사학위논문

응답 시간의 안정성 보장을 위한 공정대기열 스케줄링 기반의 플래시 메모리 제어 기법

Fair Queueing Flash Memory Control Scheme Providing Stable Response Times of Host Requests

2015 년 8 월

서울대학교 대학원
전기·컴퓨터공학부
나 안 수

초록

플래시 메모리 저장장치는 호스트의 요청으로 인해 유발되는 플래시 메모리 연산과 내부관리로 인해 발생하는 플래시 메모리 연산이 경쟁함으로 인해 호스트 요청의 응답성 및 처리율에 성능저하가 발생할 수 있다. 이러한 성능저하는 여러 SSD를 하나의 클러스터로 통합하여 활용하는 클라우드 서비스나 데이터 센터와 같은 환경에서는 치명적인 성능저하로 이어질 수 있으며, 이러한 문제를 해결하기 위한 체계적인 해결책이 필요하다.

이를 위하여 네트워크 패킷 전송 시스템에서 응답시간과 대역폭을 보장하는 목적으로 사용된 공정대기열 알고리즘을 플래시 메모리 시스템에 적용한 방식을 제안하였다. 공정대기열 알고리즘의 구현을 위해서는 이상적인 공정성 보장 모델이 필수적이다. 따라서 목표서버가 지정된 다중서버를 대상으로 하는 이상적인 공정성 보장 모델을 공정성 보장 단위와 서비스 가치 상대성의 2가지 차원으로 접근하여 통합적으로 공정성 보장 정책을 선택할 수 있게 하는 공정성 정책 분류 공간을 제안하였다. 이와 같은 이상적인 모델을 기반으로 플래시 메모리 시스템에서 경쟁하는 스트림 사이의 공정한 서비스를 제공하는 소프트웨어 계층인 공정성 보장 관리 모듈을 개발하였다. 해당 모듈은 플래시 메모리 시스템을 병렬적으로 활용하는 무순서 플래시 메모리 제어기와 연동하여 동작한다.

제안한 방식을 고해상 실시간 시뮬레이터 환경을 기반으로 구현한 결과 서비스 시간의 관점에서 스트림 사이에 공정성을 보장함을 확인할 수 있었다. 또한 새로운 스트림이 경쟁에 참여하였을 때에 해당 스트림의 응답시간을 확인하는 실험을 통해 호스트 요청의 응답시간 및 처리율의 안정성 향상을 확인할 수 있었다.

주요어 : 플래시 메모리 기반 저장장치, 공정대기열 알고리즘, QoS(Quality of Service), FTL(Flash Translation Layer), 플래시 메모리 제어기

학번 : 2013-23113

목차

초록 i

목차 iii

그림 목차 vi

수식 목차 viii

표 목차 ix

제 1 장 서론 1

1.1 연구 동기 1

1.2 연구 내용 4

1.3 논문의 구성 5

제 2 장 배경 지식 7

2.1 플래시 메모리 7

2.1.1 NAND 플래시 메모리 7

2.1.2 NAND 플래시 메모리 시스템 구조 11

2.1.3 NAND 플래시 메모리 연산 12

2.1.4 쓰레기 수집 기법 14

2.2 플래시 메모리 제어기 15

2.2.1 무순서 플래시 메모리 제어기..... 18

2.2.2 무순서 플래시 메모리기의 고해상 실시간 시뮬레이션.. 19

2.3 공정대기열 알고리즘 21
2.3.1 가중치 공정대기열 알고리즘 21
2.3.2 다중 서버를 대상으로 하는 공정대기열 알고리즘 25
2.3.3 CPU 스케줄러에서의 공정스케줄링 알고리즘 29
2.3.4 플래시 메모리 관련 공정대기열 알고리즘 30

제 3 장 목표서버가 지정된 다중서버 환경에서의 이상적인 공정성 모델 32
3.1 목표서버가 지정된 다중서버 환경 32
3.1.1 목표서버가 지정된 다중서버 환경의 특징 34
3.2 목표서버가 지정된 다중서버 환경에서의 이상적인 공정성 모델 37
3.2.1 공정성 모델의 요구사항 37
3.2.2 공정성 모델 및 공정성 모델 분류 공간 38

제 4 장 목표서버가 지정된 다중서버 환경에서의 공정성 보장 기법 45
4.1 목표서버가 지정된 다중서버 환경에서의 공정성 보장 기법 45
4.1.1 크레디트와 서버요금을 통한 공정성 보장 관리 기법... 45
4.1.2 공정성 보장 정책에 따른 크레디트와 서버요금의 관리 47
4.1.3 공정성 보장 정책에 따른 크레디트와 서버요금 관리의 구현 51

제 5 장 공정대기열 스케줄링 기반 플래시 메모리 제어 기법 55
5.1 공정대기열 스케줄링 기반 플래시 메모리 제어 기법 55

5.1.1 플래시 메모리 시스템의 특징 55
5.1.2 공정대기열 스케줄링 기반 플래시 메모리
제어 기법의 요구사항 57
5.1.3 공정대기열 스케줄링 기반 플래시 메모리 제어 계층 .. 58
5.2 공정대기열 플래시 제어 계층의 구현 59
5.2.1 공정대기열 플래시 제어 계층의 구조 59
5.2.2 공정대기열 플래시 제어 계층의
공정성 관리 기법의 구현 61

제 6 장 실험 및 결과 **63**
6.1 실험 환경 63
6.2 공정성 보장 여부 확인 실험 64
6.2.1 균등한 경쟁에서의 공정성 보장 확인 64
6.2.2 서로 다른 경쟁에서의 공정성 보장 확인 67
6.2.3 서로 다른 가중치를 갖는 경우 공정성 보장 확인 71
6.2.4 공유자원을 포함한 경우 공정성 보장 확인 74
6.3 응답성 보장 여부 확인 실험 76
6.3.1 새로운 스트림의 경쟁 참여시 응답성 보장 여부 확인 76

제 7 장 결론 **78**

참고문헌 **82**

Abstract **86**

그림 목차

그림 1 플래시 메모리 시스템의 구조 11
그림 2 SLC, MLC에서의 읽기, 쓰기 연산의
지연시간 분포 예시 13
그림 3 플래시 읽기, 쓰기, 지우기 연산의 시간 특성 예시 13
그림 4 플래시 메모리 제어기 연결 구조 및 연산 단계 16
그림 5 순차 실행 모델의 수행 예시 17
그림 6 중첩 실행 모델의 수행 예시 17
그림 7 무순서 중첩 실행 모델의 수행 예시 17
그림 8 고해상 실시간 시뮬레이터와 하드웨어의
응답시간 비교 결과 19
그림 9 N×N 링크 연결에서의 공정대기열 스케줄링 기법 26
그림 10 일반적인 다중서버 환경에서의 스트림과 서버 관계 32
그림 11 목표서버가 지정된 다중서버 환경의
스트림과 서버 관계 예시 33
그림 12 전체 시스템에 백로그를 가진 스트림에 대한
공정성 보장 근거 35
그림 13 단일 서버에서 백로그를 가진 스트림에 대한
공정성 보장 근거 36
그림 14 공정성 모델의 분류 공간 39
그림 15 GPS-DP 및 각 분류 방식 별 서비스양 흐름 예시 ....,,,,,,,,,,,,, 42
그림 16 매개변수를 통한 공정성 모델 분류공간의 관리 43

그림 17 공정성 보장 정책에 따른 크레디트의
생성과 소비 흐름 예시 50
그림 18 공정대기열 플래시 제어 계층의 구조 60
그림 19 균등한 경쟁에서의 공정성 보장 실험 결과 66
그림 20 서로 다른 경쟁에서의 공정성 보장 실험 결과 (1) 68
그림 21 서로 다른 경쟁에서의 공정성 보장 실험 결과 (2) 70
그림 22 서로 다른 경쟁에서의 공정성 보장 실험 결과 (3) 71
그림 23 서로 다른 가중치를 갖는 경우 공정성 보장 실험 결과 (1) 72
그림 24 서로 다른 가중치를 갖는 경우 공정성 보장 실험 결과 (2) 73
그림 25 공유자원을 포함한 경우 공정성 보장 실험 결과 75
그림 26 요청 발생 시간에 따른 응답시간의 분포 76
그림 27 시간에 따른 스트림별 시스템 서비스 시간의 분배 77

수식 목차

수식 1 WFQ의 가상시간 관리 23
수식 2 WFQ의 시작시점 및 종료시점 관리 24
수식 3 DRFQ에서의 가상시간, 가상시작시점, 가상종료시점 관리 28
수식 4 매개변수를 통한 서비스 보장율 및 서버 가치의 산출 44
수식 5 시스템 단위의 공정성 보장 및 고정적 요금 부여시
서버가격 및 크레디트 증가량의 결정 52
수식 6 시스템 단위의 공정성 보장 및 가변적 요금 부여시
서버가격 및 크레디트 증가량의 결정 52
수식 7 서버 단위의 공정성 보장 및 고정적 요금 부여시
서버가격 및 크레디트 증가량의 결정 53
수식 8 서버 단위의 공정성 보장 및 가변적 요금 부여시
서버가격 및 크레디트 증가량의 결정 53

표 목차

표 1 WFQ의 가상시간 관리 시 사용된 표기법 22
표 2 DRFQ에서 가상시간 관리 시 사용된 표기법 27
표 3 공정성 보장 정책에 따른 크레디트와 서버 요금의 관리 49
표 4 크레디트 및 서버요금 관리 시 사용된 표기법 51
표 5 고해상 실시간 시뮬레이터의 환경설정 변수 63
표 6 스트림 별 버스 및 칩에 대한 서비스 시간 표 64
표 7 서로 다른 경쟁 실험 3의 스트림 별 백로그된 서버 설정 70
표 8 서로 다른 가중치 실험 2의 스트림 별 백로그된 서버 설정 74

제 1 장 서론

1.1 연구 동기

플래시 메모리(Flash memory)는 기계적인 방식으로 동작하는 하드디스크(Hard-disk)와 달리 전자적인 방식으로 동작함으로써 빠른 임의 접근 속도, 낮은 전력 소비, 충격과 진동에 대한 내구성 및 연산의 높은 병렬성 등 다양한 장점을 보유하고 있다. 이와 같은 장점을 기반으로 플래시 메모리는 스마트폰과 같은 이동형 장치의 저장 매체로 주로 사용되어 왔다. 또한 플래시 메모리의 제조 공정의 세밀화와 하나의 트랜지스터에 여러 비트의 정보를 저장하는 MLC(Multi-Level Cell) 기술 및 기존의 2차원 배열 구성을 3차원으로 적층함으로써 집적도를 향상시킨 3D NAND 플래시 기술의 도입에 힘입어 플래시 메모리의 용량 대비 가격은 점차 하락하는 추세를 보이고 있다 [1]. 이를 통해 하드디스크와 비교하여 성능과 용량 대비 가격 측면에서의 높은 경쟁력을 확보한 플래시 메모리는 PC와 노트북에 사용되는 SSD(Solid State Drive)의 형태로 클라이언트 시장을 빠르게 잠식하고 있다. 또한 클라우드 서비스의 발전과 사회 관계망 서비스의 보급 등을 기반으로 성장하고 있는 서버 시장에서도 빠른 응답시간과 큰 입출력 대역폭에 대한 요구사항을 만족시키는 플래시 메모리 기반 저장장치가 도입되면서 다양한 응용분야에서 그 성장세를 보이고 있다.

수많은 사용자의 요청을 처리하는 클라우드 서버 및 데이터 센터에서는 사용자의 원활한 서비스 활용을 위하여 다양한 QoS(Quality of Service)를 필수적으로 보장해야 한다. QoS의 항목에는 서비스의 사

용가능성 보장, 높은 수준의 보안 제공, 빠른 평균 응답시간 및 최대 응답시간 제한 그리고 서비스 처리량 등이 있다 [2]. 위의 항목 중 응답시간과 서비스 처리량과 관련된 부분은 저장장치의 성능 특성과 밀접한 관계가 있으며, 따라서 서버시장에서 플래시 메모리 기반 저장장치의 성장을 위해서는 응답시간과 서비스 처리량에 대해 적합한 성능 특성을 보장하는 것이 필수적이다.

플래시 메모리는 하드디스크와는 달리 덮어쓰기가 불가능한 특성을 가지고 있으며, 이러한 특성에 대응하기 위하여 FTL(Flash Translation Layer)이라 불리는 별도의 소프트웨어 계층을 필요로 한다. 해당 소프트웨어 계층은 논리적인 섹터 주소를 물리적인 플래시 메모리의 주소와 대응시키는 기능을 제공한다. FTL은 새로운 쓰기 연산이 요청되었을 때에 기존의 데이터가 저장되어 있는 물리주소의 값을 변경하지 않고, 새로운 물리주소에 쓰기 연산을 수행한 후에 사상 정보를 변환하는 방식으로 동작한다. 이와 같은 동작으로 인해 플래시 메모리 저장장치에서는 무효화된 영역을 다시 쓰기 가능한 영역으로 환원시켜주는 쓰레기 수집(Garbage Collection) 기능이 필수적으로 수반되어야 하며, 이와 같은 쓰레기 수집 기능은 저장장치의 응답시간과 데이터 전송률에 큰 변동을 일으키는 원인이 되고 있다 [3].

이와 같은 응답시간과 데이터 전송률의 변동은 모바일 기기나 PC와 같이 소규모의 플래시 메모리 저장장치를 쓰는 환경에서 사용자 경험에 부정적인 영향을 미칠 수 있다. 또한 여러 SSD를 하나의 클러스터로 통합하여 서비스를 하는 경우 전체적인 성능 저하의 주된 요인이 될 수 있다. 해당 시스템에서 한 요청이 완료되는 시점은 각 SSD에 분배된 하위요청 중 가장 늦은 시점에 완료되는 하위요청에 의해 결정되게 된다. 이때 쓰레기 수집 기능이 수행된 SSD가 있을

경우 해당 요청의 완료시점은 쓰레기 수집 기능으로 인해 동작이 지연된 SSD에 의해 결정되게 되며, 따라서 클러스터에 포함된 SSD의 개수가 증가할수록 더 높은 확률로 성능 저하가 발생하게 된다 [4]. 이와 같이 쓰레기 수집 등의 내부 관리 동작으로 인한 응답성 및 처리율의 성능저하는 특정 시스템에서 치명적인 성능저하로 이어질 수 있다.

이를 해결하기 위하여 실제 플래시 메모리 기반 저장장치를 개발하는 산업계에서는 호스트의 요청에 의해 유발되는 호스트 요청과 플래시 메모리 내부 관리 요청으로 요청의 종류를 분류하여 각 요청 종류별 발생 횟수를 FTL단에서 조절하는 등의 접근을 취하고 있다. 그러나 해당 접근 방식은 복잡한 플래시 메모리 시스템의 구성을 반영하여 동작하는데 한계가 있으며, 특정 요청의 조합에 대해서는 목표로 하는 호스트 요청의 응답시간을 일시적으로 보장하지 못하는 경우가 발생할 수 있다. 예를 들어 플래시 메모리 *칩 0*에 다수의 읽기 요청이 발생한 이후 플래시 메모리 *칩 1*에 호스트에 의한 읽기 요청이 발생하고, 연이어 *칩 0*에 플래시 쓰레기 수집 기능이 동작을 시작한 경우, *칩 1*에서 동작할 수 있는 읽기 동작이 있음에도 불구하고 호스트 요청과 쓰레기 수집 기능이 *칩 0*에서 경쟁을 함으로써 호스트 동작의 전체적인 응답성이 현저히 떨어질 수 있다. 이러한 문제를 해결하기 위하여 각 칩별로 요청의 종류에 따른 횟수를 관리하는 등 더 복잡한 접근방식을 취하는 것도 가능할 수 있으나, 해당 접근 방식을 취한 경우에도 전체적인 호스트 요청의 응답성을 보장하는 것에 대한 문제가 남게 된다. 또한 플래시 시스템의 구성이 변화함에 따른 고려사항들의 변화는 저장장치 개발의 복잡도를 증가시키는 원인이 될 수 있다.

이에 본 연구에서는 응답시간 및 처리율을 보장하기 위하여 네트워크 분야에서 활발하게 연구된 공정대기열 알고리즘 방식을 플래시 메모리 시스템에 도입하여 쓰레기 수집 기능 등 플래시 내부 관리 동작으로 인한 호스트 응답성 및 처리율의 저하 문제에 대한 체계적이고 효과적인 처리방식을 제안한다.

1.2 연구 내용

본 논문에서는 플래시 메모리 기반 저장장치의 응답시간을 보장하기 위한 공정대기열 기반의 플래시 메모리 제어 기법을 제안한다.

제안하는 기법의 공정성을 검증하기 위한 비교대상으로서 목표서버(Destination server)가 지정된 다중서버(Multi-server)환경에서의 이상적인 공정성 모델을 제안한다. 또한 목표서버가 지정된 다중서버 환경의 특성을 고려하여 해당 공정성 모델을 공정성 보장 관리 단위와 서비스 가치 상대성의 2가지 차원으로 나누어 분류한다. 제안하는 공정대기열 기반의 플래시 메모리 제어 기법은 이상적인 모델을 근사하는 접근을 취하며, 해당 모델을 분류한 2차원의 공간에서 특정 지점을 선택할 수 있도록 2가지 매개변수를 제공함으로써 본 제어기법의 사용자가 적합한 공정성 보장 기법을 선정할 수 있도록 한다.

또한 제안하는 기법은 공정성 보장 기능의 계산 복잡도를 고려하고 활용 환경에 따른 유연성을 보장하기 위하여 소프트웨어 계층에서 구현한다. 따라서 하드웨어 계층에서 동작하는 플래시 메모리 제어기(Flash memory controller)의 동작 특성이 중요한 역할을 담당한다. 이에 본 연구진에서 개발한 무순서 플래시 메모리 제어기를 활용함으로써 플래시 메모리 시스템의 활용도를 높인다 [5].

이어 제안한 기법을 본 연구진에서 개발한 고해상 실시간 플래시 메모리 시뮬레이터를 활용한 실험을 통해 동작을 확인한다 [6]. 해당 시뮬레이터는 플래시 메모리 제어기를 포함한 실제 플래시 메모리 시스템을 마이크로초 단위의 오차 내에서 모사하는 실시간 시뮬레이터이다. 실험은 여러 스트림이 경쟁할 경우에 공정성 보장의 여부를 확인하는 실험과, 제안하는 플래시 메모리 제어 방식을 호스트 요청과 플래시 메모리 내부 관리 요청의 경쟁했을 때에 호스트 요청의 응답성의 안정성을 보장할 수 있는지의 여부를 확인하는 실험으로 나누어 진행함으로써, 공정성 및 응답성의 안정성을 보장하는 플랫폼으로써의 실효성을 확인한다.

1.3 논문의 구성

본 논문의 구성은 다음과 같다.

2장에서는 배경지식으로 플래시 메모리의 특성과 플래시 메모리 시스템의 구조, 그리고 플래시 메모리 연산 등 플래시 메모리 시스템에 대해 설명한다. 이후에는 본 연구진에서 개발한 무순서 플래시 제어기의 설계와 구현에 대하여 설명한다. 그리고 실험 환경으로 활용 될 해당 플래시 제어기를 모사한 고해상 실시간 시뮬레이터에 대해 설명한다. 이어 네트워크 분야의 패킷 처리 방식에서 연구된 공정대기열 알고리즘에 대한 선행연구에 대해 단일서버 환경에 대해 다룬 가중치 공정대기열 알고리즘의 연구와 다중 서버를 대상으로 하는 다양한 공정 대기열 알고리즘에 대해 살펴본다. 이어서 CPU 스케줄링에서의 공정대기열 알고리즘에 대해 살펴본다. 마지막으로 플래시 메모리 저장장치와 관련된 공정대기열 알고리즘을 살펴본다.

3장에서는 목표서버가 지정된 다중서버환경에서의 이상적인 공정성

모델을 제안하고 해당 모델을 분류한 후 각 분류의 특성을 설명한다.

4장에서는 3장에서 제안한 이상적인 공정성 모델을 근사한 목표서버가 지정된 다중서버 환경에서의 공정성 보장 기법을 제안한다.

5장에서는 4장에서 제안한 공정성 보장 기법을 플래시 시스템에 적용한 공정대기열 스케줄링 기반 플래시 메모리 제안 기법을 제안한다.

6장에서는 플래시 메모리 제어기의 실시간 고해상 시뮬레이터 환경에서의 실험결과를 통해 제안한 기법의 실효성을 확인한다.

마지막으로 7장에서 결론을 맺는다.

제 2 장 배경지식

2.1 플래시 메모리

플래시 메모리는 EEPROM(Electrically Erasable Programmable Read-Only Memory)의 일종으로 전원이 차단된 이후에도 저장된 정보가 유지되는 비휘발성 메모리(Non-volatile memory)이다. 플래시 메모리는 크게 NOR 플래시 메모리와 NAND 플래시 메모리로 구분할 수 있다. NOR 플래시 메모리는 바이트 단위의 빠른 임의 읽기가 가능하기 때문에 내장형 시스템의 코드 저장 용도로 사용되고 있다. NAND 플래시 메모리는 바이트 단위의 임의 읽기가 불가능하고 더 큰 단위인 페이지(Page) 단위의 임의 읽기만을 제공하는 제한점이 있으나, NOR 플래시 메모리 대비 저렴한 단위 용량 당 가격과 빠른 쓰기 속도를 기반으로 내장형 시스템 및 다양한 컴퓨터 시스템에서 저장장치로 사용되고 있다. 본 논문에서는 NAND 플래시 메모리에 대해 초점을 맞추어 연구를 진행하며, 앞으로 언급되는 플래시 메모리는 NAND 플래시 메모리를 지칭한다.

2.1.1 NAND 플래시 메모리

NAND 플래시 메모리는 빠른 임의 접근 속도, 낮은 전력 소비, 충격과 진동에 대한 내구성 및 연산의 높은 병렬성 등 다양한 장점을 보유하고 있어 내장형 시스템 및 이동형 장치의 저장 매체로 주로 사용되어 왔다. 또한 플래시 메모리의 용량 대비 가격이 하락하면서 하

드디스크와 비교하여 성능과 용량 대비 가격 측면에서의 높은 경쟁력을 확보한 플래시 메모리는 PC와 노트북에 사용되는 SSD의 형태로 클라이언트 시장을 빠르게 잠식하고 있다. 또한 클라우드 서비스의 발전과 사회 관계망 서비스의 보급 등을 통해 성장하고 있는 서버 시장에도 빠른 응답시간과 큰 입출력 대역폭에 대한 요구사항을 만족시키는 플래시 메모리 기반 저장장치가 도입되고 있다.

그러나 플래시 메모리는 덮어쓰기가 불가능한 연산 특성과, 연산 단위의 비대칭성, 그리고 불량블록(Bad block)의 발생 및 비트반전 오류 등 다양한 제약을 가지고 있어 여러 종류의 하드웨어 및 소프트웨어 관리 계층이 필요하다.

페이지와 페이지들의 집합인 블록(Block) 단위로 구성되는 NAND 플래시 메모리는 하드디스크와 달리 동일한 물리 주소에 덮어쓰기를 할 수 없고 블록 단위로 동작하는 지우기 연산이 이뤄 진 후에 쓰기 연산이 가능하다. 쓰기 연산 또한 바이트 단위의 쓰기가 불가능하며 페이지 단위의 연산만을 지원한다. 따라서 플래시 메모리는 FTL이라는 소프트웨어 모듈을 통해 논리적 주소와 물리적 주소의 사상 정보를 관리함으로써 사용자에게 논리적으로 연속적인 주소를 갖고, 임의 쓰기가 가능한 기존의 저장장치와 동일한 동작을 제공한다.

FTL은 사상 단위를 기준으로 분류 했을 때 블록 사상 기법, 페이지 사상 기법, 하이브리드 사상 기법으로 분류할 수 있다 [7]. 페이지 사상 기법은 크기가 작은 호스트 요청에 대해서도 높은 성능을 보인다는 장점이 있지만, 전체 사상 정보의 크기가 비대하다는 단점이 있다. 반대로 블록 사상 기법은 전체 사상 정보의 크기가 작지만, 작은 크기의 호스트 요청에는 성능이 취약하다는 단점이 있다. 하이브리드 사상 기법은 페이지 사상과 블록 사상 기법의 문제점을 개선하기 위하여 제안된 기법으로 사상 정보를 효율적으로 관리함과 동시에 작은

크기의 호스트 요청에도 빠른 성능을 보일 수 있으나, 해당 FTL의 구현 복잡도가 상대적으로 높다는 단점이 있다.

이러한 사상정보는 전원이 차단된 이후에도 저장장치로서 동작할 수 있도록 플래시 메모리에 저장되어야 하며, 해당 사상정보의 전원 오류 등의 오류 상황에 의한 손상에 대처하기 위하여 체크포인트 등의 추가적인 정보들을 플래시 메모리에 저장하여 관리하여야 한다. 이와 같이 호스트에 의해 유입된 사용자 데이터 이외에 플래시 메모리 저장장치의 동작을 위하여 내부적으로 관리되는 데이터들을 속성정보(Metadata)라 부른다. 이러한 속성정보의 관리는 플래시 메모리 연산이 필수적으로 수반되며 해당 플래시 메모리 연산이 호스트 요청에 의해 유발된 플래시 메모리 연산과 경쟁함으로써 호스트 요청의 성능을 저해하는 원인이 될 수 있다.

호스트의 쓰기 연산이 수행될 때에 플래시 메모리 내부의 사상정보를 갱신하면서 더 이상 호스트에게 노출되지 않는 영역인 무효화된 공간이 발생하게 된다. 무효화된 공간이 증가하게 되면 저장장치 내부에 물리적으로 기록 가능한 공간이 감소하기 때문에 해당 공간을 사용 가능한 공간으로 환원해야 한다. 이 동작을 쓰레기 수집(Garbage collection)이라고 부른다. 호스트의 요청과 쓰레기 수집 작업은 모두 플래시 메모리 연산을 유발하기 때문에 쓰레기 수집 작업이 활성화 된 경우 저장장치의 전체적인 성능 저하가 발생하게 된다 [3].

또한 플래시 메모리는 가격경쟁력을 갖추기 위한 수율 향상의 방편으로 생산과정에서 불량블록을 허용하고 있다 [8]. 이와 같은 불량블록은 초기불량블록이라고 불린다. 또한 생산과정에서 정상적으로 동작했던 정상블록도 사용 과정에서 반복적인 쓰기 연산과 지우기 연산을 거치면서 해당 연산을 완료하지 못하는 경우가 발생하는데, 이와

같은 현상이 발생한 블록을 동작중 불량블록이라고 한다. 따라서 플래시 메모리 저장장치는 이러한 불량블록을 관리하는 기능을 제공해야만 한다. 또한 쓰기와 지우기 연산의 횟수가 증가함에 따라 동작중 불량블록이 발생할 확률이 증가하므로, 안정적인 저장장치의 동작을 위하여 정상블록들에 대한 쓰기 및 지우기 횟수의 균등화 기능을 추가적으로 제공해야 한다.

쓰기 연산이 정상적으로 완료된 데이터의 경우에도 다양한 원인에 의해 비트 반전 오류로 인한 데이터 손상이 발생할 수 있다. 이와 같은 오류에 대응하기 위하여 플래시 메모리는 ECC(Error Correction Code)기법을 통해 특정 개수 이하의 비트 반전 오류를 정정하는 기능을 제공한다. 따라서 해당 비트 반전 오류가 정정기능의 한계를 넘어서지 않도록 데이터를 관리하는 기법이 필요하다. 읽기 연산이 수행 될 때, 대상 페이지에 인접한 곳에 위치한 페이지들이 받는 전기적 영향으로 인해 발생하는 비트 반전 현상을 읽기 교란(Read disturbance)이라고 부른다. 이러한 읽기 교란 현상으로 인한 데이터 손상을 방지하기 위하여 특정 블록에 수행된 읽기 연산의 횟수를 관리하여 특정 횟수를 넘어설 경우 해당 블록 전체의 데이터를 읽어 새로운 블록으로 복사함으로써 오류 정정 기능 허용 한도를 넘어서는 것을 방지한다 [9].

이와 같이 다양한 내부 관리 기능이 동작하는 플래시 메모리 저장장치는 내부 관리 기능으로 인해 유발되는 플래시 메모리 요청이 호스트 요청이 유발시키는 플래시 메모리 요청과 경쟁함에 따라 호스트 요청의 응답성 및 처리율의 저하가 발생할 수 있다. 따라서 호스트 요청과 내부 관리 요청의 경쟁을 관리하여 호스트 요청의 응답성 및 처리율을 보장하기 위한 체계적인 관리 기법이 필요하다.

2.1.2 NAND 플래시 메모리 시스템 구조

플래시 메모리 시스템에서는 호스트에 의한 읽기 및 쓰기 요청을 사상정보 관리를 포함한 내부 정책에 따라 플래시 메모리의 읽기, 쓰기, 지우기 연산으로 전환하는 FTL 모듈이 핵심적인 기능을 제공한다. 또한 FTL 모듈이 생성하는 플래시 메모리 연산들을 실제 플래시 메모리 칩 인터페이스에 호환되도록 변환하는 기능을 제공하는 모듈을 플래시 메모리 제어기라고 부른다. 플래시 메모리는 빠른 응답시간 및 높은 대역폭을 제공하기 위하여 다수의 버스와 다수의 플래시 칩을 병렬적으로 활용하는 구조를 갖는 것이 일반적이며 해당 구조는 그림 1과 같다 [5]. 이때 데이터 전송을 담당하는 버스를 다수의 플래시 메모리 칩이 공유하는 구조를 통해 하드웨어 구현 시 중요한 자원인 핀(Pin)을 효과적으로 활용하도록 한다. 따라서 FTL은 다수의 버스와 칩이 병렬적으로 동작 할 수 있도록 플래시 요청을 분배하여 생성하는 기능을 담당해야 하며, 플래시 메모리 제어기 역시 FTL이 생성한 다수의 플래시 연산이 병렬적으로 처리 될 수 있도록 플래시 요청을 효율적으로 스케줄링 하는 것이 필요하다.

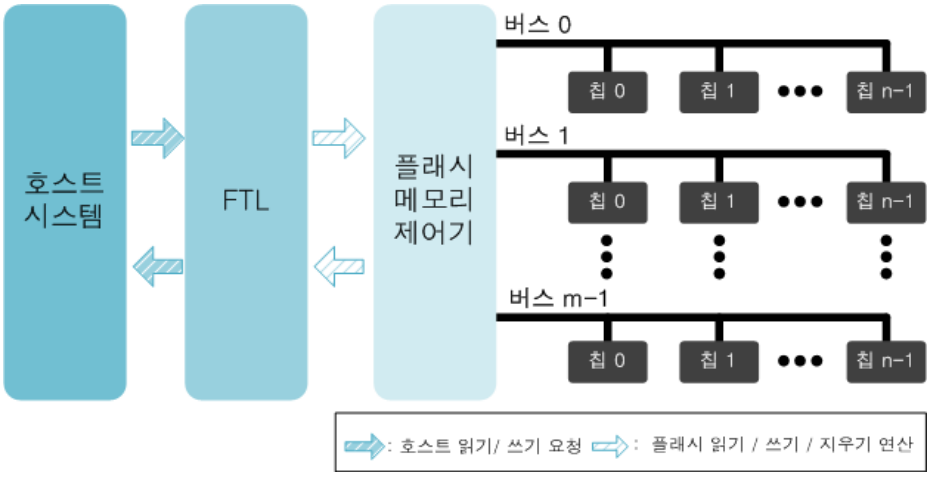

그림 1 플래시 메모리 시스템의 구조 [5]

2.1.3 NAND 플래시 메모리 연산

플래시 메모리 연산의 종류에는 읽기 연산과 쓰기 연산 그리고 지우기 연산이 있으며 각 연산의 동작 단위가 서로 다른 특성을 갖는다. 읽기 연산과 쓰기 연산은 페이지 단위로 동작하며, 지우기 연산은 페이지의 집합인 블록 단위로 동작한다. 또한 쓰기 연산은 지우기 연산이 선행된 이후에만 동작이 가능한 제약이 있다. 더 나아가 플래시 시스템의 자원인 버스와 칩을 사용하는 데 있어 각 연산이 사용하는 자원의 종류와 그 순서가 상이하다.

먼저 지우기 연산은 데이터의 이동 없이 칩 내부에 있는 특정 블록의 데이터를 지우는 동작만을 수행하므로 버스 동작 없이 칩 연산만으로 동작한다.

읽기 연산의 경우, 플래시 메모리 내부에 저장되어 있는 페이지의 데이터를 칩 내부의 페이지 버퍼(Page buffer)에 옮기는 칩 연산이 먼저 수행되어야 하며, 해당 연산이 종료된 이후에 버스를 통해 FTL에 데이터를 전송하게 된다.

쓰기 연산의 경우, 버스를 통해 칩 내부의 페이지 버퍼에 FTL에 의해 전달된 데이터를 옮기는 작업이 선행되어야 한다. 이후 해당 데이터를 페이지 버퍼에서 플래시 메모리 페이지에 옮기는 쓰기 연산이 칩에서 수행되어야 한다.

이러한 자원 사용의 선행성(Precedence) 이외에도 플래시 메모리 연산은 서로 수행 시간이 상이하다는 특성이 있다.

먼저 한 개의 플래시 메모리 셀에 여러 비트를 저장하는 MLC(Multi Level Cell)기법의 사용 여부에 의해 읽기 및 쓰기 연산의 지연시간 분포의 특성이 달라진다. 하나의 셀에 하나의 비트만을

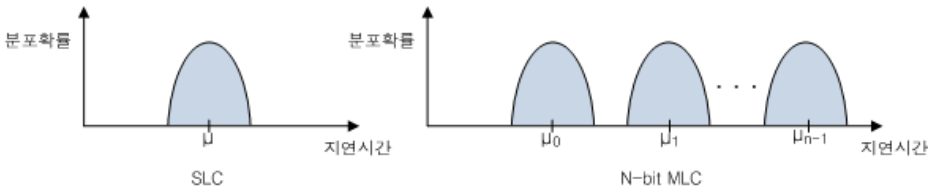

그림 2 SLC, MLC에서의 읽기, 쓰기 연산의 지연시간 분포 예시 [5]

저장하는 SLC(Single Level Cell)기법의 경우 읽기 연산과 쓰기 연산 모두 각각 하나의 특정 평균값에 집중된 분포를 보인다. 임의의 N개의 비트를 하나의 셀에 저장하는 N-bit MLC기법의 경우 한 셀에 할당된 다수의 페이지가 저장된 데이터를 읽는데 필요한 내부 정보 확인 동작의 횟수가 각각 다르며, N개의 서로 다른 평균값에 집중된 분포를 보인다 [10]. 해당 지연시간 분포의 예시는 그림 2와 같다 [5].

또한 전체적인 동작 속도가 읽기 연산이 쓰기 연산에 비해 빠르고, 쓰기 연산이 지우기 연산에 비해 빠른 특성을 보인다. 하나의 플래시 메모리 셀에 2개의 비트를 표현하는 2-bit MLC에서 사용자 데이터를

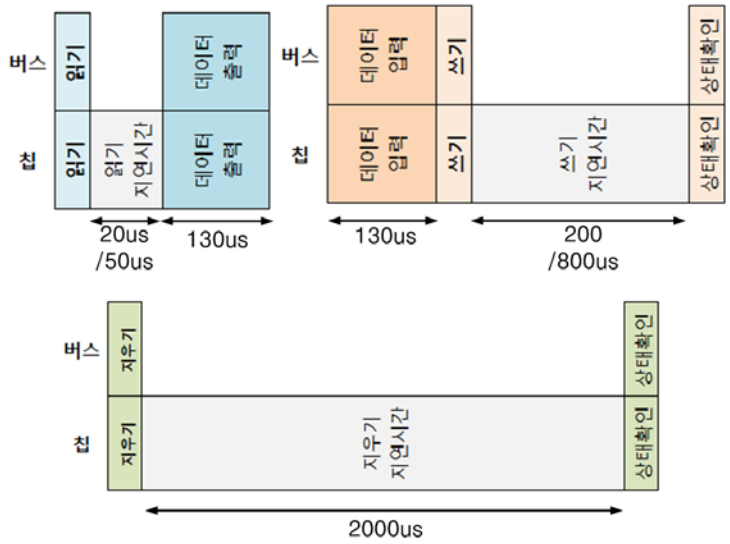

그림 3 플래시 읽기, 쓰기, 지우기 연산의 시간 특성 예시 [6]

위한 4KB의 데이터 영역과 플래시 메모리 시스템의 관리를 위한 218byte의 여유(Spare) 영역을 하나의 페이지 단위로 삼고, 버스 전송률이 33MB/s인 특정 플래시 메모리 시스템에서 각 연산의 동작 시간 특성은 그림 3과 같다 [6]. 이때 읽기, 쓰기, 지우기 연산의 칩 동작을 유발시키는 명령 및 상태확인 명령은 수백 나노초 이하의 시간 내에 종료된다.

2.1.4 쓰레기 수집 기법

논리적인 주소와 물리적인 주소의 사상정보를 관리하며 동작하는 플래시 메모리는 공정과정 및 동작과정에서 무효화된 블록이 생성되어 해당 영역을 관리하는 쓰레기 수집 기능의 제공이 필수적이다.

쓰레기 수집 기법 정책의 주요 정책은 해당 동작의 수행 시점, 처리할 무효화된 영역을 포함하는 블록 수, 동작을 수행할 블록의 선정, 유효한 영역을 관리하는 기법 등이 포함된다.

해당 동작의 수행 시점은 주로 수위표(Watermark)를 활용하는 방식으로, 물리적으로 쓰기가 가능한 자유 블록(Free block)의 개수가 특정 한계점(Threshold)이하로 줄어들 때에 쓰레기 수집 기법을 통해 무효화된 영역을 포함하는 블록에서 유효한 페이지만을 복사해 옮김으로써 자유 블록을 확보하게 된다 [11].

쓰레기 수집 동작을 수행할 블록을 선정하는 방식의 예로는, 무효화된 영역의 비율이 가장 높은 블록을 선정하는 탐욕 알고리즘(Greedy Algorithm)이 존재한다.

또한 유효한 영역을 관리할 때에 자주 접근되는 데이터와 그렇지 않은 데이터를 분류하여 쓰레기 수집 동작의 효율성을 높이는 동적

데이터 집단화(Dynamic Data Clustering) 기법도 존재한다 [12].

이와 같은 쓰레기 수집 기법은 플래시 메모리 저장장치의 동작을 유지하는데 필수적이나, 호스트가 요청하는 플래시 메모리의 읽기 및 쓰기 연산과 경쟁하면서 호스트 요청의 응답성 및 처리율을 저해하는 주된 원인이 되고 있다.

2.2 플래시 메모리 제어기

플래시 메모리 제어기는 FTL에 의해 생성된 연산을 플래시 메모리 칩 인터페이스에 맞추어 수행을 관리하는 기능과, 여러 버스와 칩으로 구성된 플래시 메모리 시스템의 자원들을 병렬적으로 활용하는 기능을 담당한다. 이때 연산의 수행 제약에 따라 다수의 버스와 칩을 병렬적으로 활용하는 방식은 다양하게 분류될 수 있다.

먼저 플래시 메모리 버스에는 전용의 데이터 버스가 있으며 각 버스는 병렬적으로 동작이 가능하다. 또한 하나의 버스를 공유하는 다수의 칩은 공유하는 자원 없이 독립적으로 칩 연산을 수행한다. 따라서 다수의 플래시 메모리 칩을 병렬적으로 활용하는 문제는 여러 칩이 공유하는 데이터 전송을 위하여 공유하는 하나의 버스를 어떻게 공유할 지의 문제로 귀결된다.

플래시 메모리 버스를 공유하는 플래시 메모리 칩들을 활용하는 방법은 순차 실행, 순서 중첩 실행, 무순서 중첩 실행 모델로 정의된다. 단일 플래시 연산은 개시단계와 종료단계, 그리고 그 사이의 지연시간으로 나눌 수 있으며 그 구성은 그림 4와 같다 [5]. 이때 개시단계와 종료단계는 버스와 칩이 모두 사용되는 단계이며, 지연시간은 버스 자원의 활용 없이 칩에서 독립적으로 칩 연산을 수행하는 시간이

그림 4 플래시 메모리 제어기 연결 구조 및 연산 단계 [5]

다. 따라서 여러 플래시 메모리 연산이 병렬적으로 실행되는 경우에도 공유 자원인 플래시 버스는 한 번에 하나의 연산이 이용해야 하지만 플래시 칩이 내부적으로 동작중인 지연시간 기간에는 플래시 연산이 진행 중인 칩을 제외하면 플래시 버스와 다른 플래시 메모리 칩들을 모두 활용 할 수 있다.

이러한 특성을 고려할 때에, 칩의 병렬성을 고려하지 않고 개시단계, 지연시간, 종료단계를 묶어서 하나의 원자적인 연산으로 삼아 순차적으로 동작시키는 방식을 순차적 실행 모델이라고 부른다. 또한 연산의 수행 순서를 지키며 개시단계와 종료단계 사이의 지연시간에서 발생하는 병렬성을 활용하는 방식을 순서 중첩 모델이라 부른다. 더 나아가 연산의 수행 순서와 상관없이 개시단계와 종료단계 사이의 지연시간에서 발생하는 병렬성을 최대한 활용하는 방식을 무순서 중첩 실행 모델이라 부른다. 각 동작의 동작 예시는 그림 5, 6, 7과 같다 [5].

그림 5 순차 실행 모델의 수행 예시 [5]

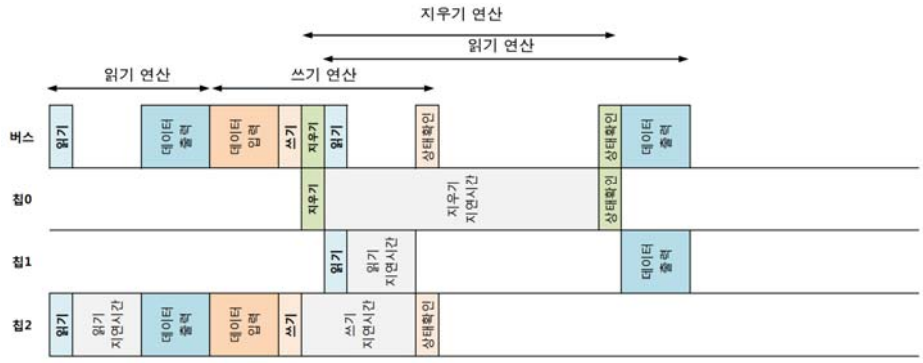

그림 6 중첩 실행 모델의 수행 예시 [5]

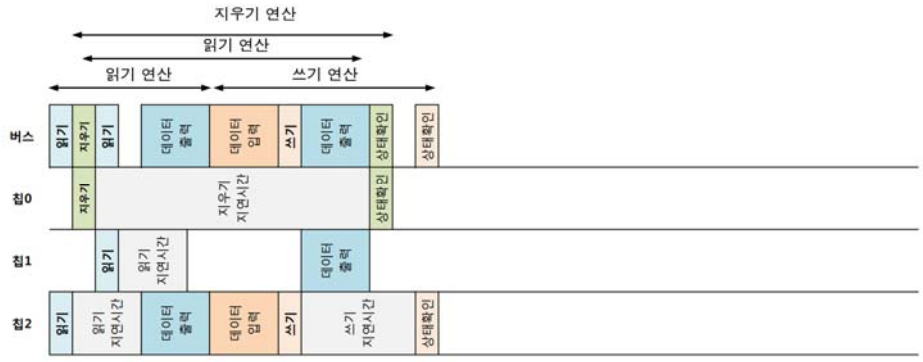

그림 7 무순서 중첩 실행 모델의 수행 예시 [5]

2.2.1 무순서 플래시 메모리 제어기

무순서 플래시 메모리 제어기는 다음과 같은 특성을 지닌다.

1) 플래시 연산 수행의 병렬성 확보를 위하여 플래시 요청들의 처리 순서에 대한 제약을 최소화하였다.

2) 플래시 연산의 효율적인 스케줄링을 위하여 버스와 칩의 상태가 가장 잘 관찰되는 하드웨어에서 스케줄링을 전담하게 하였다.

3) FTL과 플래시 메모리 제어기 사이의 인터페이스를 패킷 인터페이스로 정의하여 패킷 인터페이스를 따르는 어떠한 FTL도 무순서 플래시 메모리 제어기를 이용할 수 있게 하였다.

이와 같은 특성을 기반으로 본 연구에서 제안하는 공정성 보장을 위한 플래시 메모리 제어 기법은 하드웨어 계층에서 동작하는 무순서 플래시 메모리 제어기의 병렬성을 활용하면서, 복잡한 공정성 보장 알고리즘을 제공하기 위하여 소프트웨어 계층에 위치하도록 한다. 또한 패킷 인터페이스를 활용함으로써 무순서 플래시 메모리 제어기와의 높은 연동성 및 모듈성을 확보한다.

무순서 플래시 메모리는 입력받은 요청에 대한 순서관계에 대한 제약이 없다. 따라서 무순서 플래시 메모리를 대상으로 하는 요청에 대해 공정성 보장을 목표로 수락제어(Admission control)기능을 제공하는 공정성 보장 계층은 동작의 정확도(Correctness)를 여전히 보장한다.

2.2.2 무순서 플래시 메모리 제어기의 고해상 실시간 시뮬레이션

위와 같은 동작 특성을 보이는 무순서 플래시 메모리 제어기는 하드웨어 모듈로 동작하기 때문에 초기 개발 단계에서 활용하는데 적합하지 않다. 이에 본 연구진에서는 해당 플래시 메모리 제어기와 동일한 동작 특성을 보이는, 고해상 실시간 시뮬레이터를 개발하였다 [6].

해당 시뮬레이터는 칩 수준의 모든 동작을 '사건'으로 다루며, 각 사건의 발생과 해당 사건의 발생으로 인한 시간의 지연 및 사건 발생으로 인한 후속 사건의 인과적 발생을 모델링한 이산사건구동형 시뮬레이션으로 설계되었다.

또한 각 칩수준의 동작과 버스수준의 동작의 시간 특성을 실제 하드웨어에서 측정된 값을 기반으로 모델링함으로써 그 신뢰성을 높였고, 실험을 통해 동일 시간 특성을 기반으로 할 경우 동일한 스케줄링 결과 및 동작 시간 특성을 보임을 검증하였다. 해당 연구의 실험 결과는 그림 8과 같다 [6].

본 연구의 실험은 위 연구를 통해 정확성이 검증된 고해상 실시간

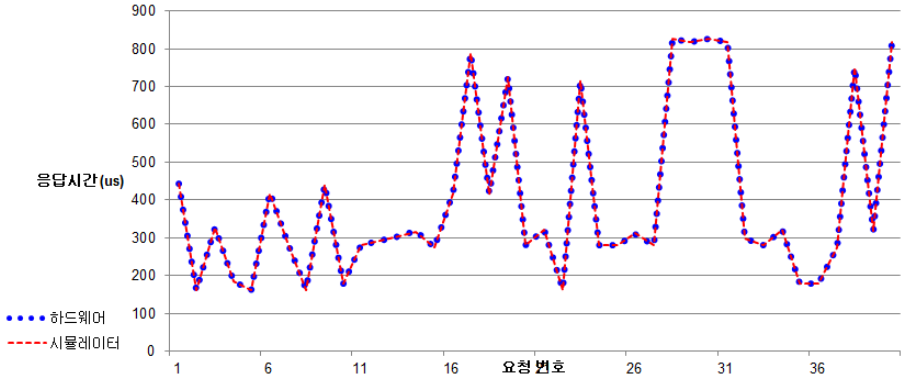

그림 8 고해상 실시간 시뮬레이터와 하드웨어의 응답시간 비교 결과 [6]

시뮬레이터를 기반으로 실험을 진행하였다.

2.3 공정대기열 알고리즘

공정대기열 알고리즘은 네트워크 패킷 처리 시스템에서 대역폭 및 응답시간 관점의 공정 서비스를 제공하는 기법이다. 공정대기열 알고리즘은 과거에 받은 서비스를 기억하지 않고, 오직 현재 서버에 처리할 패킷이 남아있는 스트림이 보유한 가중치에 비례하여 서버의 처리 용량을 분배한다. 이때 처리할 패킷이 남아있는 스트림을 백로그(Backlog)가 있는 스트림이라 부르며, 해당 서버도 백로그를 지닌다고 표현한다. 공정대기열 알고리즘은 백로그가 있는 한 서버가 유휴 상태를 갖지 않는 작업량보장성(Work-conserving)의 특성을 갖는다 [13].

공정대기열 알고리즘이 제안되기 전 초창기의 패킷 처리 시스템에서는 먼저 도착한 패킷이 먼저 서비스를 받는 FCFS(First-Come, First-Served)방식을 취했다. 그러나 이 경우 특정 스트림이 과도한 패킷을 전송 할 때에 해당 스트림 이외의 스트림이 전송한 패킷이 서비스를 받지 못하는 기아(Starvation)현상이 발생 할 수 있다. 이에 각 스트림 마다 별도의 대기열을 할당 받아 각 대기열에 적재되어 있는 패킷을 순환 순차 방식으로 서비스를 제공함으로써 공정성을 보장하는 방식이 제안되었다 [14].

2.3.1 가중치 공정대기열 알고리즘

초창기의 공정대기열 알고리즘은 각 스트림이 제공받는 패킷의 서비스 횟수만을 고려하기 때문에 패킷 당 처리해야 할 데이터의 양이 다양한 경우 공정성을 보장할 수 없다. 이에 단위 시간당 하나의 비트 단위로 각 스트림에 순환 순차 방식으로 서비스를 제공하는 이상

적인 모델인 GPS(Generalized Processor Sharing)모델을 정의하고, 해당 모델의 처리를 근사하여 서로 다른 데이터양을 보유한 패킷 사이의 공정성을 보장하는 가중치 공정대기열 알고리즘인 WFQ(Weighted Fair Queueing)방식이 제안되었다 [15]. 또한 가중치 공정대기열 방식과 트래픽 성형 방식인 리키 버킷(Leaky Bucket) 방식을 결합하여 대역폭 및 응답시간의 최악의 성능(Worst-Case performance)을 일정 수준 이내로 보장하는 방식이 제안되었다 [16].

가중치 공정대기열 알고리즘은 GPS를 모사함으로서 패킷 처리의 종료시점을 계산하고, 가장 빠른 종료시점을 갖는 처리를 우선적으로 수행하는 스케줄링 정책을 갖는다. 해당 종료시점을 산출하는 것은 높은 수준의 계산복잡도가 요구되며, 해당 계산복잡도를 줄이기 위한 다양한 노력이 시도되었다.

먼저 실제 시간의 흐름에 따른 변화하는 가상시간(Virtual time)을 시스템적으로 유지함으로써 종료시점을 계산하는 방식이 제안되었다 [16]. 해당 가상시간은 모든 스트림의 대기열이 비었을 때에 0으로 초기화 된다. 그리고 각 스트림은 가상시간 당 처리하는 서비스의 비율

표기법	설명
$v(t)$	실제시간 t에서의 가상 시간
t_j	j번째 이벤트가 발생한 실제 시간
B_j	j-1번째 이벤트 발생과 j번째 이벤트 발생 사이에 백로그를 지닌 스트림의 집합
W_i	스트림 i가 갖는 가중치
$a_{i,k}$	스트림 i의 k번째 패킷의 실제 도착 시간
$S_{i,k}$	스트림 i의 k번째 패킷의 가상 시작시점
$F_{i,k}$	스트림 i의 k번째 패킷의 가상 종료시점
$L_{i,k}$	스트림 i의 k번째 패킷의 데이터 전송량

표 1 WFQ의 가상시간 관리 시 사용된 표기법 [16]

$$v(0) = 0$$

$$v(t_{j-1} + \tau) = v(t_{j-1}) + \frac{\tau}{\sum_{i \in B_j} W_i}$$

$$\tau \leq t_j - t_{j-1}$$

수식 1 WFQ의 가상시간 관리 [16]

을 자신의 가중치에 비례하여 받게 된다. 또한 가상시간이 증가하는 속도는 백로그가 있는 스트림의 가중치의 합에 반비례한 속도를 갖는다. 이에 백로그가 있는 스트림의 가중치가 클 때에는 가상시간이 천천히 흐르고 단위 가상시간 당 처리율을 일정하게 유지함으로써 단위 시간에 한 스트림이 받는 실제 서비스의 양이 줄어들게 된다. 반대로 백로그가 있는 스트림의 가중치가 작을 때에는 가상시간이 빨리 흐름으로써 단위 시간에 한 스트림이 받는 실제 서비스의 양이 증가하게 된다. 이로써 경쟁이 증가함에 따라 한 스트림이 실제 서비스 받는 시간이 감소하고, 경쟁이 감소함에 따라 한 스트림이 실제 서비스 받는 시간이 증가하는 경향을 반영한다.

GPS상에서 특정 패킷의 서비스가 시작되거나 종료되는 것을 하나의 사건이라고 했을 때, 표 1과 같은 표기법을 사용할 경우 *j-1*번째 이벤트와 *j*번째 이벤트 사이에 가상시간 *v*의 관리 공식은 수식 1과 같다 [16]. 이때. 경쟁에 참여하지 않던 스트림에 다시 백로그가 생길 경우, 해당 스트림의 시작시간을 현재 시간에 대응되는 가상시간으로 설정함으로써 새롭게 백로그된 스트림이 이전에 서비스 받지 않았던 기간의 보상으로 서버를 상당 시간 동안 독점하는 것을 방지한다. 이는 스트림 *i*의 *k*번째 패킷의 시작시점 S를 실제 패킷의 도달시점인 $a_{k,i}$와 *k-1*번째 패킷의 종료시점 F 중 더 큰 값으로 선정하여 수식 2와 같은 방식으로 관리된다 [16].

$$S_{i,k} = \max(F_{i,k-1}, v(a_{i,k}))$$
$$F_{i,k} = S_{i,k} + \frac{L_{i,k}}{W_i}$$

수식 2 WFQ의 시작시점 및 종료시점 관리 [16]

그러나 해당 *N*개의 스트림이 있을 때에 해당 가상시간을 관리하기 위한 복잡도가 *O(N)*으로 다소 높다는 단점이 있다.

이에 가상시간을 근사화 시켜 구현 복잡도를 개선하기 위하여 SCFQ(Self-clocked fair queueing) 및 SFS(Star-time fair queueing) 등의 방식이 제안되었다 [17][18]. SCFQ에서는 실제 처리되는 데이터의 양을 기반으로 가상시간을 현재 서비스 중인 세션의 가상 종료시점으로 근사하였다. SFS에서는 가상시간을 현재 서버 내의 세션 중 최소의 가상 시작시간으로 근사하였고, 스케줄링 방식도 가상 종료시점을 기준으로 하지 않고 가상 시작시점이 가장 빠른 것을 선정하는 방식을 취했다.

또한 가중치 공정대기열 방식이 가상 종료시점만을 고려함으로써 발생하는 오차를 줄이기 위하여 가상 시작시점을 지난 패킷을 스케줄링의 대상으로 삼는 WF^2Q(Worst-case Fair Weighted Fair Queueing)방식이 제안되었다 [19]. 그러나 해당 방식은 가상시간 유지 복잡도가 *O(N)*으로 다소 높다는 단점이 있다.

이와 같이 네트워크 패킷 처리 시스템에서 다양한 연구가 진행되었으나 해당 연구들은 단일 서버 환경을 주요 연구 환경으로 삼았다. 이는 패킷을 전송하는 기기인 라우터에서의 동작이 단일 서버에서의 동작과 유사하며, 서버에 의해 처리된 이후에 하나의 출력 대기열을 통해 전달된 패킷이 병렬적으로 여러 링크에 전송되는 구조가 그 원인으로 생각된다.

2.3.2 다중 서버를 대상으로 하는 공정대기열 알고리즘

단일 서버 상황을 고려한 대부분의 연구 외에 다수의 링크와 링크 사이를 연결하는 라우터에서 출력 링크 단에서의 경쟁을 출력 링크 단위로 관리하는 다중 서버 대상의 공정대기열 방식이 제안되었다 [20].

해당 방식은 입력 링크와 출력 링크의 조합 개수만큼의 대기열을 유지하면서 각 출력링크가 독자적인 가상시간 v를 입력 링크의 개수만큼 관리하며 다음과 같은 요청, 승인, 전송의 절차를 반복한다. 각 입력 링크는 백로그가 있는 유휴상태의 출력 링크에 패킷 전송을 요청한다. 각 출력 링크는 요청을 받은 입력 링크 중 가장 작은 가상시간을 갖는 입력 링크를 선정하여 승인 신호를 보낸다. 이때 여러 출력 링크에게 동시에 승인 신호를 받은 입력 링크는 대기 시간이 가장 오래된 패킷을 선정하여 해당 패킷만을 전송 대상으로 삼는다. SFQ방식과 유사한 접근 방식을 통해 각 출력링크 단위에서 해당 출력링크에 백로그가 있는 입력링크에 대해 대역폭의 공정성을 보장하게 되며, 모든 입력링크가 모든 출력링크에 대해 백로그가 있는 경우 전체적인 공정성도 보장되게 된다.

그러나 이러한 요청, 승인, 전송 절차가 반복될 때에 출력링크가 전송을 승인한 입력링크가 해당 출력링크를 전송 대상으로 선정하지 않았을 때에, 다른 링크에 의해 전송이 수행될 수 있었음에도 불구하고 해당 전송이 수행되지 못하는 문제가 발생한다. 이는 공정성의 보장에는 문제가 없으나, 전체 서비스의 성능 저하를 유발한다.

또한 지역적인 가상시간만을 관리하므로 출력 링크사이에 백로그된 입력 링크의 불균형이 발생했을 때에 전체적인 공정성을 보장할 수

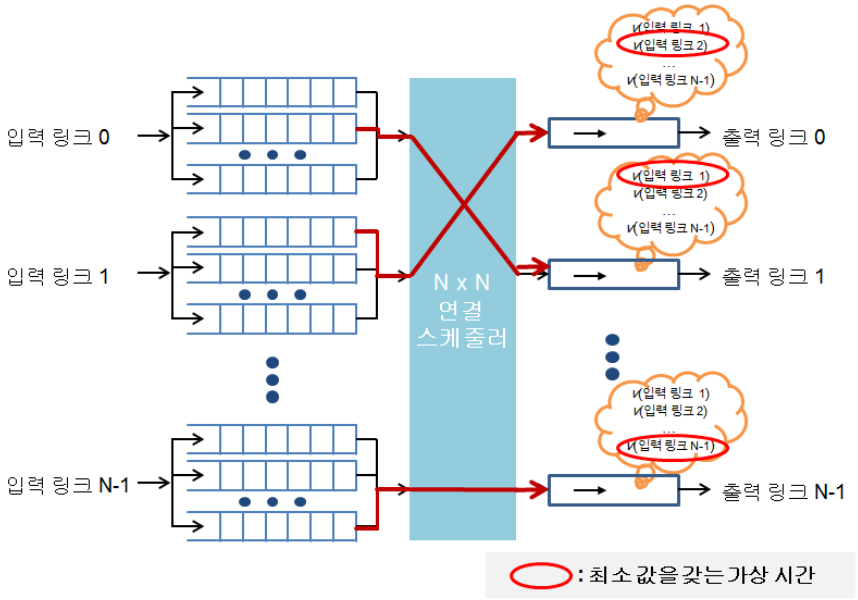

그림 9 N×N 링크 연결에서의 공정대기열 스케줄링 기법 [20]

없는 단점이 있다. 해당 기법의 개략적인 구성은 그림 9와 같다 [20].

위의 방식과 유사한 접근으로 다수의 출력 링크에 대한 공정성 보장을 목표로 하되 다수의 출력 링크를 하나의 서버로 모델링하여 GPS를 근사하는 방식을 취한 링크 병합 공정대기열 알고리즘 방식도 제안되었다 [21]. 다만 이 방식은 입력 링크에서 전송된 패킷이 임의의 출력링크를 선정할 수 있는 상황으로 환경을 한정하고 있으며, 플래시 메모리와 같이 목표서버가 지정된 다중서버 환경에서 발생할 수 있는 서버 사이의 경쟁 불균형이 발생하는 문제는 다루지 않고 있다.

또한 네트워크 패킷 전송의 중간자 역할로서 방화벽, 부하 분산 기능, NAT(network address translation) 등의 기능을 담당하는 하는 미들장비(Middle box)에서 다수의 서버를 관리하는 DRFQ(Dominant Resource Fair Queueing)기법이 제안되었다 [22]. 미들장비의 주된 서버는 CPU와 NIC(Network interface controller)로서 하나의 미들박스 안에는 소수의 서버만이 존재한다.

표기법	설명
$v(t)$	실제시간 t에서의 가상 시간
$p_{i,k}$	스트림 i의 k번째 패킷
$a_{i,k}$	스트림 i의 k번째 패킷의 실제 도착 시간
$s_{k,i,j}$	스트림 i의 k번째 패킷의 자원 j에서의 처리 시간
W_i	스트림 i가 갖는 가중치
$S(p_{i,k})$	스트림 i의 k번째 패킷의 시스템 가상시작시점
$F(p_{i,k})$	스트림 i의 k번째 패킷의 시스템 가상종료시점
$S(p,j)$	패킷 p의 자원 j에서의 가상시작시점
$F(p,j)$	패킷 p의 자원 j에서의 가상종료시점
$P(t)$	시간 t에서 서비스 받고 있는 패킷의 집합

표 2 DRFQ에서 가상시간 관리 시 사용된 표기법 [22]

DRFQ방식은 서로 다른 종류의 서버를 요구하는 수행 사이에 공정성을 제공하는 DRF(Dominant resource fairness) 기법을 기반으로 한다.

DRF기법은 주로 데이터 센터나 클라우드 서버에서 활용되는 기법으로서 해당 작업이 요청하는 수행에 필요한 서비스양을 미리 알고 있어야 수행이 가능하다. 제공되는 각 서버종류의 총 서비스양 대비 요청하는 각 서버종류의 서비스양의 비율 중 더 높은 비율의 요청을 해당 서버의 가중치 적용 대상으로 삼는 방식이다. 위와 같은 방식을 통해 여러 종류의 서버에 서로 다른 요청량을 갖는 다수의 작업이 경쟁을 할 때에 공정한 서비스 제공을 보장하게 된다 [23].

데이터 센터와 클라우드 서비스에서의 DRF기법은 제공되는 자원의 양의 총 개수가 경쟁하는 작업의 개수보다 더 많기 때문에 동시에 모든 작업이 자원을 나누어 받을 수 있다. 그러나 패킷 전송 처리의 경우 한 번에 사용할 수 있는 자원의 개수보다 처리해야 할 작업의 개수가 월등히 많은 경우가 대다수이다. 따라서 DRFQ방식은 시분할 다중방식(Time-division multiplexing)의 접근을 취한다.

$$v(0) = 0$$
$$v(t) = \begin{cases} \max\{S(p,j) | p \in P(t)\} & \text{if } (P(t) \neq \varnothing) \\ 0 & \text{if } (P(t) = \varnothing) \end{cases}$$

$$S(p_{i,k}) = \max(v(a_{i,k}), F(P_{i,k-1}))$$
$$F(p_{i,k}) = S(p_{i,k}) + \frac{\max_j \{s_{k,i,j}\}}{W_i}$$

수식 3 DRFQ에서의 가상시간, 가상시작시점, 가상종료시점 관리 [22]

DRFQ방식은 SFQ방식과 유사한 접근방식을 취하되, 우세한 자원에 대해 가중치를 부여받는 효과를 얻기 위하여 가상시간을 적용할 때에 가장 많은 시간동안 서비스 받은 자원에서의 서비스 시간을 활용하여 가상종료시점을 결정한다. 이러한 동작은 표 2와 같은 표기법을 따를 때, 수식 3을 기반으로 동작한다 [22].

DRFQ와 같은 접근을 플래시 메모리 시스템에 적용했을 때에는 다음과 같은 한계가 존재한다.

1) 플래시 메모리는 미들장비에 비해 활용하는 서버의 개수가 더 많아 해당 방식을 유지하는 복잡도가 높다.

2) 다수의 칩에 의해 공유되는 버스에 대한 경쟁을 반영할 수 없다. 플래시 메모리의 요청은 버스와 칩을 동시에 사용하는 개시, 종료 구간과 칩만을 사용하는 구간으로 나누어져 있어 항상 특정 칩에서의 처리시간이 버스에서의 처리시간보다 길다. 따라서 여러 칩에 의해 공유되는 버스에서의 상대적 서비스 요구량이 높음에도 불구하고 독립적인 칩에서의 서비스를 기준으로 공정성을 보장받게 된다.

3) 플래시 메모리는 연산의 종류에 의해 활용하는 서버의 종류와

순서가 결정된다. 이와 같은 선행성에 대한 고려가 해당 방식에서는 결여되어 있다.

지금까지 살펴본 본 것과 같이 다중 서버 환경에서의 공정대기열 알고리즘의 연구결과는 하나의 공유되는 서버를 여러 독립적인 서버가 공유하는 동시에, 각 서비스의 목표서버가 지정되어 있는 플래시 메모리 시스템의 독특한 환경으로 인해 효율적인 적용이 어려우며 플래시 시스템 환경에 특화된 공정성 보장 방식의 연구가 필요하다.

2.3.3 CPU 스케줄러에서의 공정스케줄링 알고리즘

CPU 스케줄러에서도 공정대기열 알고리즘과 유사한 스케줄링 정책들이 연구되었다.

먼저 단일 CPU를 대상으로 하는 스트라이드 스케줄링(Stride scheduling)방식이 제안되었다. 해당 스케줄링 방식은 가중치에 해당하는 티켓(Ticket)을 프로세스 별로 부여하고, 티켓에 반비례하도록 각 프로세스에 스트라이드를 부여한다. 그리고 해당 프로세스가 CPU를 단위시간 동안 점유할 때마다 스트라이드 값만큼 가상시간의 역할을 하는 패스(Pass)를 증가시킨다. 이 후 가장 작은 값의 패스를 소유한 프로세스에게 CPU를 할당한다. 이와 같은 접근방식을 취할 경우 경쟁하는 프로세스의 상태가 유지될 때에 특정 시간 주기를 단위로 엄밀하고 결정적인(Deterministic) 공정 스케줄링(Fair scheduling)을 지원하게 된다 [24]. 이와 같은 방식은 공정대기열 알고리즘 중 SFQ와 유사한 방식이라 생각할 수 있다.

또한 다수의 CPU로 구성된 멀티프로세서 환경에서 동작하는 스케줄러로서 리눅스 운영체제에 도입된 CFS(Completely Fair

Scheduler)가 제안되었다 [25]. 해당 스케줄링 방식은 이상적인 멀티태스킹 CPU를 가정하고 해당 CPU를 모사함으로써 멀티프로세서 환경에서의 공정성을 보장한다. 이를 위하여 운영체제는 CPU시간이 흐름에 따라 특정 비율로 증가하는 공정시계(Fair clock)를 관리한다. 해당 공정시계의 시간은 실제시간을 기다리는 프로세스의 수에 우선순위(Priority)를 반영한 값으로 나눈 만큼 증가한다. CFS는 CPU를 점유할 프로세스를 선정하기 위하여 프로세스별로 대기시간(Waiting time)이라는 값을 유지하며 최대 대기시간을 보유한 프로세스가 CPU를 점유하게 된다. 대기시간은 CPU를 점유하지 못한 경우 기다리는 프로세스의 수에 반비례하는 값만큼 증가하며, CPU를 점유할 경우 실제 시간의 값만큼 감소한다. 이를 통해 이상적인 멀티태스킹 CPU에서의 수행을 모사한다. 시간이 흐름에 따라 최대 대기시간을 보유한 프로세스가 바뀌게 되고, 새롭게 최대 대기시간을 보유한 프로세스가 CPU를 점유하게 된다. 해당 작업의 수행 시간의 단축을 위하여 수행중인 프로세스의 집합을 레드블랙트리(Red black tree)를 활용하여 대기시간에 따른 순서를 관리한다.

이와 같은 멀티프로세서에서의 CPU 점유시간에 대한 공정성 보장 방식은 선점(Preemptive)방식으로 동작하며 각 프로세스가 모든 CPU에서 처리가 가능하다는 점에서 비선점(Nonpreemptive)방식으로 동작하면서 목표서버가 지정된 플래시 메모리 시스템에서의 환경과 차이가 있다.

2.3.4 플래시 메모리 관련 공정대기열 알고리즘

여러 사용자가 제한된 수의 서버를 활용하는 경우, 각 사용자의 서버 활용에 대해 공정성을 보장하는 방식 수립이 필수적이다. 따라서 여러 호스트 요청이 경쟁하는 플래시 메모리 기반 저장장치에 대해서 공정성을 보장하고자 하는 FOG(Flash Operation Groups) 스케줄링

기법이 제안되었다 [26].

해당 기법은 SSD 내부에 다수의 대기열 구조를 제공하여 병렬적인 동작을 허용하는 NVMe 인터페이스를 기반으로 CFS 방식과 유사한 접근 방식을 취한다. 스트림 별로 버스와 칩에서 받는 서비스 시간을 기반으로 가상 시작시점을 관리한다. 또한 전체 시스템의 가상시간은 모든 스트림의 현재 요청의 가상 시작시점 중 최소값으로 선정한다. 이때 가상 종료시점은 가상 시작 시점에 버스와 칩의 수행시간의 합을 반영한 명목 수행시간을 기반으로 계산한다. 이후 해당 요청이 종료되었을 때에 실제 수행시간과 명목 수행시간의 차이를 반영하여 지속적인 공정성 보장을 제공한다.

해당 기법은 플래시 메모리 시스템을 대상으로 다수의 요청대기열을 유지함으로써 여러 스트림의 역동적인 경쟁에도 균등하면서도 효율적인 서비스시간을 보장하였다.

그러나 해당 기법은 하나의 버스에 하나의 칩만이 경쟁하는 상황을 가정하고 FTL단에서의 부하 균등화를 가정함으로써 특정 서버에서의 다수의 플래시 메모리 칩이 버스를 공유하는 구조적인 경쟁불균형과, 스트림의 목표지점이 시간에 따라 변화함에 따른 시간적 경쟁불균형에 대해서는 적실한 해결책을 제시하지는 않고 있다.

따라서 해당 연구에서의 병렬성을 보장하기 위한 기법들을 활용하되 경쟁불균형으로 인한 문제를 해결하기 위한 추가적인 연구가 필요한 상황이다.

제 3 장 목표서버가 지정된 다중서버 환경에서의 이상적인 공정성 모델

3.1 목표서버가 지정된 다중서버 환경

목표서버가 지정된 다중서버 환경은 일반적인 다중서버 환경과는 다른 특성을 갖는다. 목표서버가 지정된 다중서버 환경이란 서비스를 요청하는 스트림이 요구하는 서버가 지정되어 있는 환경을 말한다. 이때 일반적인 다중서버 환경에서는 서버의 부하를 고려하여 스트림이 요청받을 서버를 선택함으로써 부하 균등화(Load-balancing)를

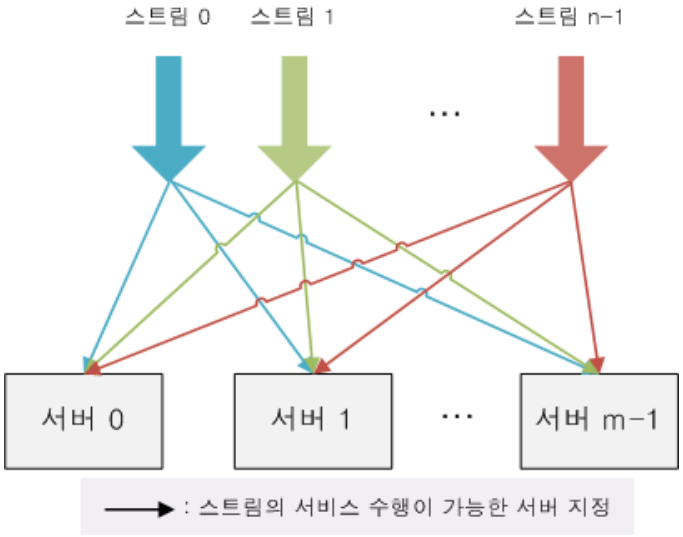

그림 10 일반적인 다중서버 환경에서의 스트림과 서버 관계

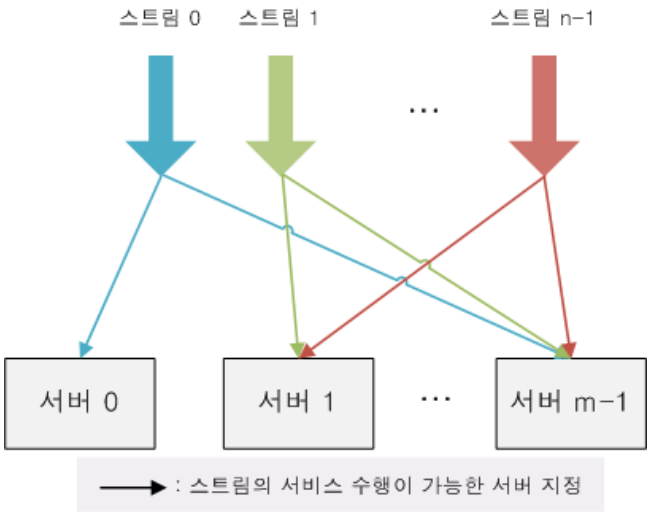

그림 11 목표서버가 지정된 다중서버 환경의 스트림과 서버 관계 예시

수행할 수 있으며 그 스트림과 수행 가능한 서버와의 관계는 그림 10과 같다. 이와 같은 부하 균등화를 통해 다중서버에서의 처리는 하나의 이상적인 서버인 GPS로 근사화가 가능하다. 그러나 그림 11과 같이 목표서버가 지정된 다중서버 환경에서는 특정 요청에 대해 수행 가능한 서버가 고정되어 있어 이와 같은 부하 균등화가 불가능하므로 일반적인 다중서버 환경에서와 같이 GPS를 이상적인 모델로 삼아 근사하는 접근 방식으로는 공정한 서비스를 제공할 수 없다.

2장에서 살펴본 다중서버환경에서의 공정대기열 알고리즘 및 멀티프로세서 스케줄러에서의 공정 스케줄링 알고리즘은 대부분 목표서버에 대한 제약이 없는 상황을 다룬다. 따라서 부하 균등화를 통한 단일 서버로서의 근사를 통해 시스템의 전체적인 요구사항에 대한 공정성을 보장이 가능하였다. 소수의 서버를 지닌 다중서버환경에서의 공정성을 다룬 DRFQ방식의 경우, 목표서버가 지정된 다중서버 환경을 다루고 있으나 하나의 작업이 소수의 모든 서버의 서비스를 요구하는 상황을 다룬다. 따라서 다수의 다중서버 환경에서 발생할 수 있는, 특

정 스트림이 특정 서버를 요구하지 않는 경우에 대한 고려가 결여되어 있다. 따라서 버스와 같은 공유서버의 경우, 스트림의 요청이 지속적으로 부여됨으로써 항상 우세한 자원으로 분류되어 버스 이외에 다수의 칩에서의 발생하는 경쟁상황을 제대로 관리할 수 없다.

플래시 메모리 시스템을 대상으로 한 FOGS방식 또한 FTL이 순차순환 방식 등의 서비스를 통해 각 서버에 대한 부화 균등화 기능을 제공하는 환경을 가정함으로써 목표서버가 지정된 다중서버의 부하 불균형 현상에 대한 해결방안을 직접적으로 제시하고 있지 않다.

따라서 목표서버가 지정된 다중서버 환경에 대한 특징에 대해 고찰하고 해당 특징을 반영하는 새로운 이상적인 모델의 도입 및 해결방안의 제시가 필요하다.

3.1.1 목표서버가 지정된 다중서버 환경의 특징

목표서버가 지정된 다중서버 환경에서, 하나의 시스템에서 경쟁하는 스트림들은 시스템 내부의 특정 서버 내부에서도 경쟁을 하게 된다. 이때 목표서버가 지정되어 있기 때문에 각 스트림이 요구하는 서버의 개수가 스트림에 따라 상이할 수 있으며, 각 서버에서 경쟁하는 스트림의 수 역시 서버에 따라 상이할 수 있다. 이러한 스트림이 요구하는 서버자원의 비대칭성과, 각 서버에 부과되는 스트림의 요구량의 비대칭성에 대한 고려가 모두 필요하다.

이때 각 스트림이 요구하는 서버의 개수의 비대칭성을 고려할 때에 각 스트림이 받는 서비스에 대한 공정성은 다음과 같은 관점에 따라 다른 정의를 지니게 된다.

1) 전체 시스템에 백로그를 가진 스트림에 대한 공정성 보장

2) 단일 서버에서 백로그를 가진 스트림에 대한 공정성 보장

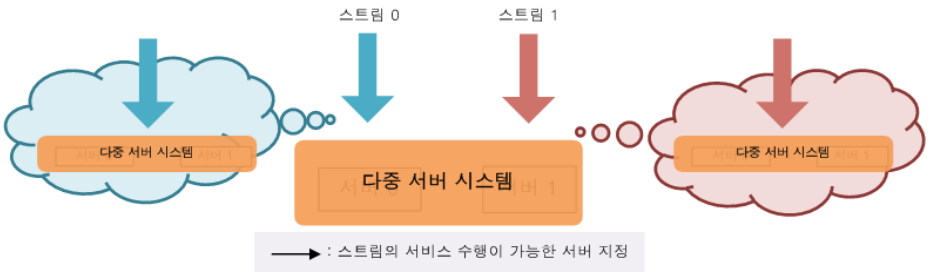

그림 12 전체 시스템에 백로그를 가진 스트림에 대한 공정성 보장 근거

전체 시스템에 백로그를 가진 스트림에 대한 공정성은 시스템 전체적인 공정성을 고려하여 하나의 서버에서 일어나는 경쟁을 관리한다. 단일 서버에서 백로그를 가진 스트림에 대한 공정성 보장은 목표서버를 가진 서버 환경의 특성을 고려하여 시스템 전체가 아닌 단일 서버에서의 공정성만을 고려하여 서버에서 일어나는 경쟁을 관리한다. 이때 단일 서버에서 백로그를 가진 스트림에 대한 공정성 보장의 관점은 모든 서버에 모든 스트림이 백로그를 가진 경우 단일 서버에서의 공정성 보장과 동일한 결과를 보이게 된다.

이와 같은 공정성 보장 관점은 각 스트림이 별도의 다중서버 시스템을 할당받은 상황을 고려할 때에 공정성의 근거가 명확해 진다.

전체 시스템에 백로그를 가진 스트림에 대한 공정성 보장의 경우, 그림 12에서 볼 수 있는 바와 같이 목표서버 지정 조합과 상관없이 해당 스트림은 전체 시스템에 대해 백로그를 갖는다. 따라서 해당 스트림은 전체 시스템의 관점에서 서버의 일정 서비스율을 보상받아야만 한다.

단일 서버에서 백로그를 가진 스트림에 대한 공정성 보장의 경우, 목표서버 지정 조합에 따라 전체 시스템에서 서비스 받는 요청의 수행이 변화하게 된다. 예를 들어 그림 13과 같이 *스트림 0*의 경우 *서*

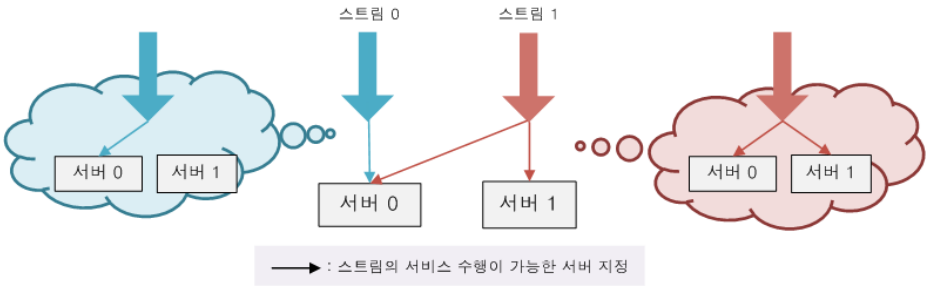

그림 13 단일 서버에서 백로그를 가진 스트림에 대한 공정성 보장 근거

*버 0*에 대해서만, *스트림 1*의 경우 *서버 0*과 *서버 1*에 백로그를 갖는 상황을 가정해보자. *스트림 0*의 경우, 모든 요청이 *서버 0*에서만 서비스 되므로 일정 시간이 지났을 때 *서버 0*에서 받는 서비스양만큼의 서비스를 제공받는다. *스트림 1*의 경우 *서버 0*과 *서버 1*으로부터 병렬적으로 서비스를 받기 때문에 일정 시간이 지났을 때 *서버 0*에서 받는 서비스양과 *서버 1*에서 받는 서비스양을 더한 값만큼의 서비스를 제공받는다. 따라서 두 스트림이 전체 시스템을 할당받았을 때를 고려하여 단위 시간동안 받는 서비스의 양은 *스트림 1*이 더 많은 양을 할당 받는 것이 공정하다. 이때 *스트림 0*과 *스트림 1*이 독립적으로 할당받은 *서버 0*에 대해 동일한 요청을 부여한다면 *서버 1*에서의 동작과 상관없이 동일한 시간에 종료하게 된다. 따라서 하나의 서버에서의 서비스 분배가 다른 서버에서의 서비스 분배와 독립적으로 관리된다면 병렬성을 고려한 다중서버에서의 공정성이 보장될 수 있다.

또한 각 서버에 부과되는 스트림의 요구량에 대한 비대칭성을 고려할 때에 각 스트림이 받는 서비스에 대한 공정성은 다음과 같은 관점에 따라 다른 정의를 지니게 된다.

1) 서비스 요구량과 무관하게 모든 서버가 동일한 서비스 가치를 보유

2) 서비스 요구량에 비례하여 각 서버가 서로 다른 서비스 가치를

보유

서비스 요구량과 무관하게 모든 서버가 동일한 서비스 가치를 보유한 경우, 시스템에서 한 서버가 활용된 실제 시간만을 기반으로 스트림 사이의 공정성이 관리되게 된다. 서비스 요구량에 비례하여 각 서버가 서로 다른 서비스 가치를 보유한 경우, 서비스 요구가 상대적으로 높은 서버에서 특정 스트림이 서비스를 받는 경우, 서비스 요구가 상대적으로 적은 서버에서 서비스를 받는 스트림보다 더 많은 서비스를 받은 스트림으로 여겨진다. 이러한 서비스 가치에 대한 상대적 관점은 DRFQ에서와 유사하게 더 많은 경쟁이 일어나는 서버에 더 높은 우선순위를 부여함으로써 공정성을 반영하는 방식이라 볼 수 있다.

공정성 보장 단위에서 바라본 2가지 관점과 서버의 서비스 가치 관리에서 바라본 2가지 관점은 서로 독립적인 접근 방식을 취한다. 따라서 각 접근을 독립적으로 조합할 경우 총 4가지 관점의 서로 다른 공정성 보장 관점이 존재할 수 있다. 서로 다른 공정성 보장 관점은 각각의 독특한 특성을 지니고 있다. 따라서 4가지 관점의 공정성 보장 방식과 각 방식의 특성을 특정 비율로 나누어 갖는 공정성 보장 방식을 통합적으로 제공할 수 있다면, 사용자는 시스템의 활용 의도에 따라 더 적합한 공정성 보장 관점을 선정할 수 있게 된다.

3.2 목표서버가 지정된 다중서버 환경에서의 이상적인 공정성 모델

3.2.1 공정성 모델의 요구사항

목표서버가 지정된 다중서버 상황에서의 이상적인 공정성 모델은

다음과 같은 요구사항을 갖는다.

1) 각 스트림이 서로 다른 서버의 집합에 백로그를 지닌 상황에서의 공정한 서비스에 대한 관점을 반영하여야 한다.

2) 각 서버가 백로그된 서로 다른 스트림의 집합을 지닌 상황에서의 공정한 서비스에 대한 관점을 반영하여야 한다.

3) 처리될 요청이 있을 때 서버는 유휴상태에 빠지지 않는다.

4) 특정 스트림이 새롭게 경쟁에 참여할 경우에 공정한 처리에 대한 관점을 반영하여야 한다.

이와 같은 요구사항을 기반으로 공정성 보장 단위와 서비스 가치 관리의 서로 다른 차원에 따라 분류된 4가지 공정성 보장 관점에 각각 대응되는 공정성 모델을 제시한다. 또한 분류된 모델을 2차원 평면으로 대응시킨 후 각 모델의 특성을 특정 비율로 반영하여 공정성을 보장하도록 하는 공정성 모델 분류 공간을 정의하여 다양한 스펙트럼을 갖는 이상적인 모델을 구성한다.

3.2.2 공정성 모델 및 공정성 모델 분류 공간

본 연구에서는 공정성 모델의 이상적인 모델로서 GPS-DM(General Processor Sharing of Destination assigned Multi-server)방식을 제안한다. GPS-DM은 단일 서버에서의 이상적인 모델인 GPS를 기반으로 구성되며 그 분류 공간은 그림 14와 같다.

시스템 단위의 백로그를 단위로 공정성을 보장하는 경우 GPS-DM은 하나의 GPS로 구성된다. 서버 단위의 백로그를 단위로 공정성을 보장하는 경우 각 서버마다 하나의 GPS를 할당하여 다수의 독립적인 GPS의 집합으로 GPS-DM을 구성한다.

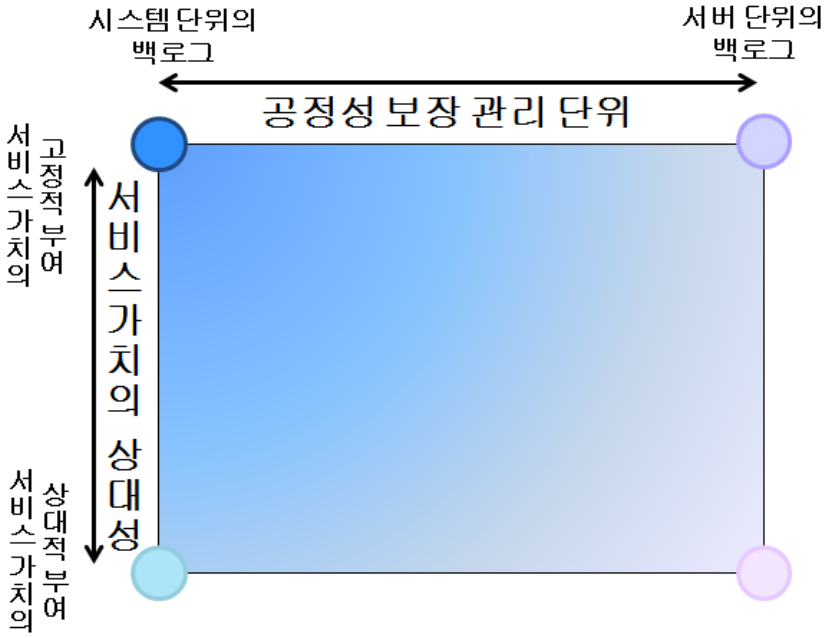

그림 14 공정성 모델의 분류 공간

서비스 가치를 고정적으로 할당하는 경우 GPS-DM에서 각 서버의 서비스양에 기여하는 비중은 모든 서버가 동일하다. 서비스 가치를 상대적으로 할당하는 경우 경쟁의 비율을 반영하여 서비스양의 기여 비중을 할당한다.

공정성 모델 1> 시스템 단위의 백로그로 공정성을 관리하며 고정적 서비스 가치를 부여받는 경우

해당 공정성 모델은 다중서버 시스템을 단일서버 시스템과 동일하게 다룬다. 다라서 단일서버에서 동작을 이상화한 GPS와 동일한 동작 특성을 보이며, 스트림이 지니는 목표 서버 집합과 각 서버에서의 스트림의 경쟁 상황과 상관없이 각 스트림에 가중치에 비례하여 전체 시스템에서의 서비스 시간을 엄밀하게 분배한다.

그러나 해당 공정성 모델은 서버 수준에서의 공정성을 고려하지 않기 때문에 스트림이 지니는 목표 서버 집합에 불균형이 발생했을 경우 더 많은 목표 서버 집합에 서비스를 요청하는 스트림이 특정 서버에서 서비스를 받지 못하는 기아현상(Starvation)이 발생 할 수 있다.

공정성 모델 2> 시스템 단위의 백로그로 공정성을 관리하며 상대적 서비스 가치를 부여받는 경우

해당 공정성 모델은 전체 시스템에서 서비스 받아야 할 서비스양을 각 서버에 분배한 후, 특정 스트림이 백로그를 갖는 서버의 개수를 고려하여 특정 서버에서의 경쟁을 관리함으로써 전체 시스템에서의 공정성을 보장한다. 이때 관리 정책은 더 많은 서버에 백로그를 지닌 요청에게 더 낮은 서비스율을 제공하는 것이다. 따라서 스트림이 지닌 백로그된 서버의 집합에 불균형이 발생할 경우에도 기아현상이 발생하지 않으며, 해당 서버에서 보장받아야 하는 서비스율을 경쟁하는 스트림 사이에서 공정하게 서비스 받을 수 있다. 또한 경쟁이 발생하지 않은 서버에 낮은 서비스가치를 부여함으로써 경쟁률이 높은 서버에서 받은 서비스양보다 더 많은 서비스를 받을 때에도 전체 서비스율에 반영되는 비율을 경쟁률이 높은 서버에서 받은 서비스율과 동일시함으로써 작업량보장성의 특징을 갖는다.

공정성 모델 3> 서버 단위의 백로그로 공정성을 관리하며 고정적 서비스 가치를 부여받는 경우

해당 공정성 모델은 서버단위로 공정성을 보장함으로서 다수의 독립적인 단일서버 환경을 다루는 것과 동일한 특성을 지닌다. 각 서버에서의 경쟁은 다른 서버에서의 경쟁에 영향을 미치지 않으며, 이는

시간적으로 백로그를 갖지 않는 스트림에 대해 보상을 해주지 않는 기존 공정대기열 알고리즘의 공정성 보장 정책과 유사하게 특정 서버에 백로그를 갖지 않는 스트림에 대해 보상을 해주지 않는 것으로 볼 수 있다.

공정성 모델 4> 서버 단위의 백로그로 공정성을 관리하며 상대적 서비스 가치를 부여받는 경우

해당 공정성 모델은 모델 3와 동일한 특성을 지닌다. 경쟁률이 더 낮은 서버의 경우 서버가 제공하는 서비스의 가치량이 더 적으나, 실제 제공하는 서비스의 양은 실제 서비스양을 스트림의 가중치에 따라 나누어 갖은 것과 같다.

위에서 살펴본 공정성 보장 모델의 GPS-DM의 구성과 서비스율의 구성의 예시는 그림 15와 같다. 그림 15는 동일한 가중치를 갖는 두 개의 스트림이 각각 1개의 서버(*서버 0*)에 백로그를 그리고 2개의 서버에(*서버 0, 서버 1*)에 백로그를 갖는 경우를 나타낸다.

이 때 서버 단위의 백로그 관리 기법에서, 여러 단일서버로 나뉘어 구성한 GPS를 하나의 GPS로 통합할 경우, 서버에서의 경쟁상황에 따라 각 스트림이 전체 시스템에서 분배받는 서비스양이 본래 의도한 1:1의 분배와는 차이가 남을 알 수 있다. 그러나 4가지 분류방식 모두, 전체시스템이 제공하는 서비스의 가치양 만큼의 서비스 가치량을 동일 시간동안 소비하고 있음을 확인할 수 있다.

이 때 공정성 보장 관리단위 관점에서 분류 공간을 대표하는 매개변수를 $\alpha(0 \leq \alpha \leq 1)$라, 서비스가치의 상대성 관점에서 분류 공간을 대표하는 매개 변수를 $\beta(0 \leq \beta \leq 1)$라고 할 때 각 매개변수가 공정성 보장 모델에 대응되는 방식은 그림 16과 같다. 이 때 스트림 i가 특

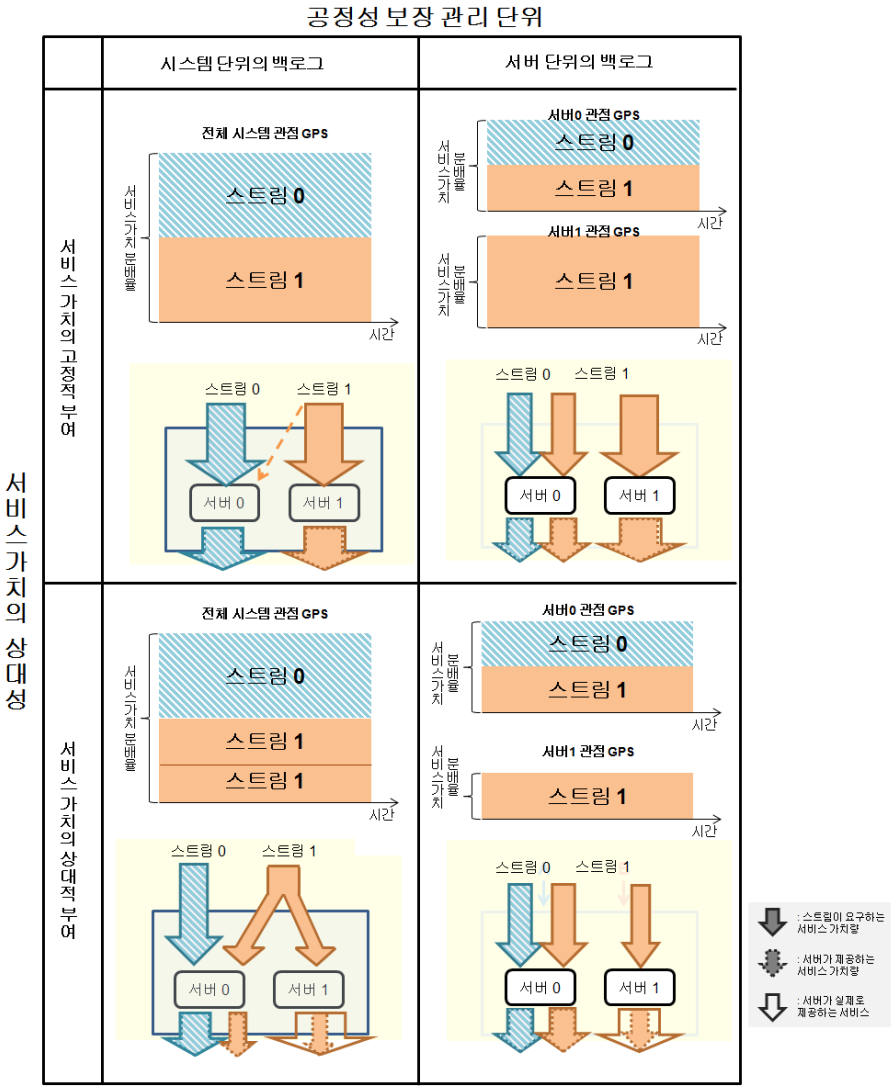

그림 15 GPS-DP 및 각 분류 방식 별 서비스양 흐름 예시

정 서버 k에서 보장받아야 하는 서비스율 *S(i,k)*과 각 서버가 제공하는 서비스의 상대적 가치 *P(i,k)*을 매개변수를 통해 산출함으로써 공정성 보장 모델의 분류공간의 특정 정책을 지원하도록 한다.

시스템 단위의 백로그를 기반으로 공정성을 관리하고 상대적 서비스 가치를 부여하는 경우의 보장 서비스율과 서비스 가치를 각각 $S_{0,0}$과 $P_{0,0}$이라 하고, 시스템 단위의 백로그를 기반으로 공정성을 관리하고 서비스 가치를 고정적으로 부여하는 기법의 보장 서비스율과 서비스 가치를 각각 $S_{0,1}$과 $P_{0,1}$, 서버 단위의 백로그를 기반으로 공정성을 관리하고 가변적 서비스 가치를 부여하는 경우의 보장 서비스율과 서비스 가치를 각각 $S_{1,0}$과 $P_{1,0}$, 그리고 서버 단위의 백로그를 기반으로 공정성을 관리하고 고정적 서비스 가치를 부여하는 경우의 보장 서비스율과 서비스 가치를 각각 $S_{1,1}$과 $P_{1,1}$이라 할 때, 매개변수를 통한 서비스 보장율 및 서버가치는 수식 4와 같이 산출된다.

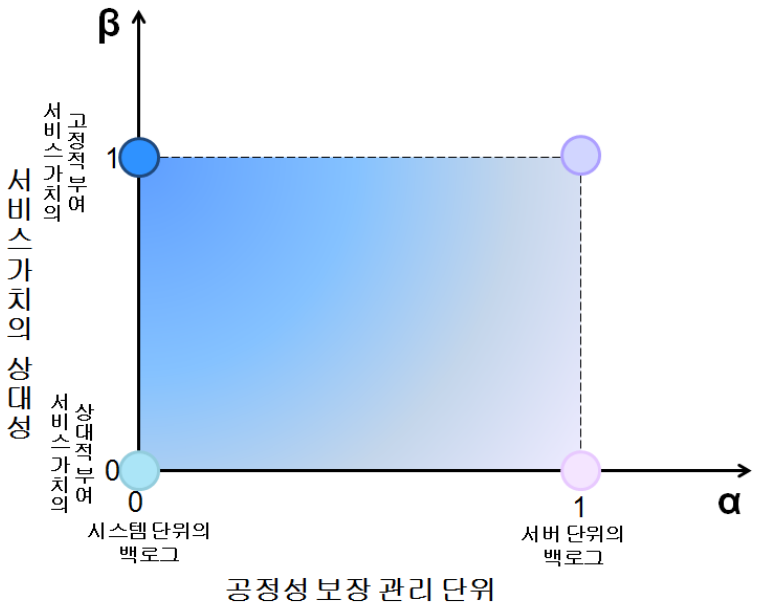

그림 16 매개변수를 통한 공정성 모델 분류공간의 관리

$$S_{\alpha,\beta}(i,k) = \{\, S_{0,0}(i,k) \times (1-\alpha) + S_{1,0}(i,k) \times \alpha \,\} \times (1-\beta) + \{\, S_{0,1}(i,k) \times (1-\alpha) + S_{1,1}(i,k) \times \alpha \,\} \times \beta$$

$$P_{\alpha,\beta}(i,k) = \{\, P_{0,0}(i,k) \times (1-\alpha) + P_{1,0}(i,k) \times \alpha \,\} \times (1-\beta) + \{\, P_{0,1}(i,k) \times (1-\alpha) + P_{1,1}(i,k) \times \alpha \,\} \times \beta$$

이와 같은 공정성 보장 모델의 분류공간을 통해 사용자는 보장하고자 하는 시스템의 특성에 따라 다양한 공정성 보장 정책 중 가장 적절한 공정성 보장 정책을 선택할 수 있으며, 모든 보장 정책은 각자 관점에서 유의미한 공정성을 보장한다.

제 4 장 목표서버가 지정된 다중서버 환경에서의 공정성 보장 기법

4.1 목표서버가 지정된 다중 서버 환경에서의 공정성 보장 기법

4장에서는 3장에서 제안한 이상적인 공정성 모델을 근사한 공정성 보장 기법을 제안한다. 해당 보장 기법은 공정성 분류 공간의 관리 정책을 하나의 통합적인 방식으로 제공하되 매개변수를 통해 사용자가 적합한 공정성 관리 정책을 채택할 수 있도록 한다. 또한 실제 시스템에 적용을 위해서 낮은 구현 복잡도로 효과적인 근사를 하는 것을 목표로 한다.

4.1.1 크레디트와 서버요금을 통한 공정성 보장 관리 기법

공정성 보장을 관리하기 위하여 사용되는 기법은 가상시간을 활용하여 이상적인 모델에서 이루어지는 처리의 가상 시작시점 및 가상 종료시점을 계산하는 방식이 주로 제안되어 왔다. 하지만 해당 방식은 스트림의 개수가 증가함에 따라 계산 복잡도가 증가한다는 단점이 있다. 또한 목표서버가 지정된 다중 서버 환경에서는 각 서버마다 부여되는 경쟁이 상이하기 때문에 각 서버에서의 가상 시작시점 및 종료 시점을 따로 관리하는 것은 서버의 개수가 증가함에 따라 그 구현

복잡도가 급격히 증가하는 결과를 낳게 된다.

또한 이상적인 공정성 모델은 독립적인 서버에서의 공정성을 보장하는 차원이 있는 반면, 전체적인 서버에서의 공정성을 보장하는 차원이 동시에 존재한다. 이를 위하여 지역적으로 공정성을 관리하는 방식과 전역적으로 공정성을 관리하는 2가지 방식을 따로 구현하여 선택하는 방식을 취할 수도 있으나, 이 경우 제안한 공정성 분류 공간 중 극단적인 정책들만을 제공하게 된다.

따라서 본 연구에서는 스트림에게 실제 이상적인 모델에서 제공받아야 하는 서비스양을 근사화한 크레디트(Credit)를 부과하고 각 서버가 제공하는 서비스양을 근사화한 요금(Price)을 각 서버에게 부과하여 전역적으로 관리하는 기법을 제안한다.

특정 서버에 백로그가 있는 경우, 공정성 관리 정책에 따라 해당 스트림의 크레디트는 단위 시간당 일정 양이 증가하게 된다. 또한 실제 서버에서 서비스를 받는 스트림은 단위 시간당 해당 서버의 요금만큼 크레디트가 감소하게 된다. 이 때 실제 서비스를 받는 스트림이 해당 서버에 백로그를 여전히 보유하고 있는 경우 스트림이 보유한 크레디트는 단위 시간당 크레디트 증가량과 서버의 요금을 차감한 만큼 변화하게 된다.

또한 새로운 스트림이 경쟁에 참여했을 때에 공정한 경쟁을 제공하기 위하여, 시간이 지남에 따라 특정 스트림이 지속적으로 크레디트를 누적하지 않도록 단위 시간당 증가되는 크레디트 양의 총합과 단위 시간당 소비되는 서버 요금의 총합이 일치하게 크레디트와 서버 요금을 관리한다.

이와 같은 접근은 CFS방식에서의 경쟁을 반영한 가상시간인 대기시간을 통해 서비스를 받지 못한 프로세서에게 이상적인 모델에서 받

았어야 하는 서비스양을 관리하고, 동시에 실제 서비스를 받는 동안 실제 받게 되는 서비스양을 가상시간에서 차감하는 방식과 유사하다. 이때 대기시간은 크레디트로, 실제 서비스를 받는 것은 서버의 요금으로 대응된다. 다만 CFS에서는 부화 균등화를 통해 모든 서버에 동일한 경쟁이 부과되므로 실제 서비스양이 하나의 기준인 실제 서비스 시간에 대응되었으나, 목표서버가 지정된 다중서버 환경에서는 서버마다 서로 다른 경쟁이 부과되므로 서버가 서로 다른 가격을 갖는 것을 허용한다.

4.1.2 공정성 보장 정책에 따른 크레디트와 서버요금의 관리

공정성 보장 정책의 적용을 위하여 다음과 같은 크레디트 및 서버 요금 관리정책을 제공한다.

1) 공정성 보장 관리 단위에 따른 크레디트 증가량 관리

크레디트의 증가량은 공정성 보장 관리 단위가 변화함에 따라 관리 정책이 달라진다.

시스템 단위의 백로그를 기반으로 공정성을 보장하는 경우, 시스템에 백로그를 가지는 스트림은 전체 시스템의 관점에서 가중치에 비례하는 서비스를 제공받아야 한다. 따라서 모든 스트림은 가중치에 비례하는 단위 시간당 크레디트 증가량을 부여받는다. 이때 동일한 가중치를 지닌 스트림은 동일한 서비스양을 보장받게 된다.

서버 단위의 백로그를 기반으로 공정성을 보장하는 경우, 특정 서버에 백로그를 지닌 스트림은 해당 서버의 관점에서 가중치에

비례하는 서비스를 제공받아야 한다. 따라서 각 스트림은 특정 서버의 요금으로부터 가중치에 비례하는 단위 시간당 크레디트 증가량을 분배받는다. 이때 하나의 크레디트가 전역적으로 관리되므로 각 서버로부터 분배받는 크레디트 증가량의 총합이 실제 크레디트 증가량으로 적용된다.

2) 서버 요금의 상대성 여부에 따른 서버요금 관리

서버의 요금은 경쟁에 따른 서비스의 상대적 가치를 고려하는지의 여부에 따라 그 관리정책이 변화하게 된다.

경쟁과 상관없이 고정적인 서버요금을 부여한 경우, 모든 서버가 동일한 요금을 보유하게 된다.

경쟁을 고려하여 상대적인 서버요금을 부여한 경우, 해당 서버에 백로그를 지닌 스트림이 요구하는 서비스 요구 양에 비례한 요금을 부여받게 된다. 이때 각 스트림이 요구하는 서비스양은 공정성 보장 관리 단위에 따라 서로 다른 특성을 갖는다.

먼저 시스템 단위의 백로그를 기반으로 공정성을 관리하는 경우 특정 스트림의 크레디트 증가량은 고정된 하나의 값을 갖는다. 따라서 해당 스트림이 백로그를 갖는 서버의 개수에 반비례한 값만큼 해당 서버에 서비스를 요구하게 되며, 이와 같이 산출된 스트림의 요구양의 총합이 해당 서버의 요금이 된다.

또한 서버 단위의 백로그를 기반으로 공정성을 관리하는 경우 특정 스트림의 크레디트 증가량은 백로그를 가진 서버에서 가중치를 반영한 요금의 총합으로 결정되게 된다. 따라서 이때 서버에 부여되는 요금은 백로그를 가진 스트림의 수에 정비례한 값을 갖게 된다.

이와 같은 관리 정책은 표 3과 같이 정리될 수 있다.

		공정성 보장 관리 단위			
		시스템 단위의 백로그		서버 단위의 백로그	
서버 요금의 상대성	고정적 서버 요금	크레디트 증가량	모든 스트림이 가중치에 비례하는 증가량 보유	크레디트 증가량	백로그를 가진 서버의 가중치를 반영한 요금의 총합
		서버 요금	모든 서버가 동일한 요금 보유	서버 요금	모든 서버가 동일한 요금 보유
	상대적 서버 요금	크레디트 증가량	모든 스트림이 가중치에 비례하는 증가량 보유	크레디트 증가량	백로그를 가진 서버의 가중치를 반영한 요금의 총합
		서버 요금	경쟁에 참여한 스트림이 요구하는 서비스양의 총합	서버 요금	경쟁에 참여한 스트림이 요구하는 서비스양의 총합

표 3 공정성 보장 정책에 따른 크레디트와 서버 요금의 관리

전역적으로 관리되는 하나의 크레디트를 통해 시스템 단위의 공정성을 제공하는 것은 자연스러운 접근 방식이다. 그러나 서버 단위의 공정성을 제공하는 것은 올바른 동작의 검증이 필요하다.

이는 크레디트의 생성과 소비의 흐름을 고려함으로써 검증이 가능하다. 단일 시간 동안 특정 서버에서 유발된 크레디트 증가량과 단일 시간 동안 해당 서버에서의 크레디트 소비량은 가중치를 고려했을 때에 같은 값을 갖는다. 따라서 짧은 시간 동안 특정 서버가 다른 서버에서 유발된 크레디트 증가량을 소비하는 경우가 발생할 수 있으나, 이와 같은 소비의 불균형은 반대 상황에서도 발생이 가능하다. 따라

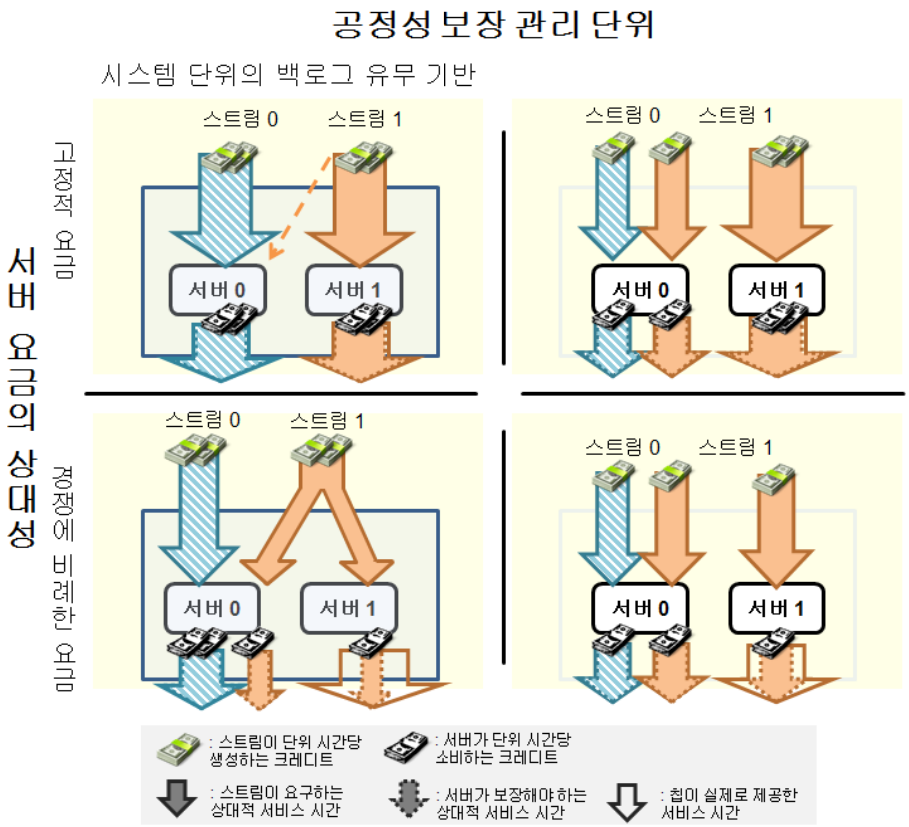

그림 17 공정성 보장 정책에 따른 크레디트의 생성과 소비 흐름 예시

서 서버와 스트림에 대해 순차 순환 방식 등의 관리 기법을 통해 동일한 크레디트 값에 대해 균등한 기회를 제공하면 전체적인 서비스 분배를 균등하게 제공할 수 있다.

이와 같은 크레디트 생성과 소비의 흐름의 예시는 그림 17에서 살펴볼 수 있다. 해당 예시는 2개의 서버가 있는 시스템에 *스트림 0*은 *서버 0*에만 백로그를, *스트림 1*은 *서버 0*과 *서버 1*모두에 백로그를 갖는 환경에서 크레디트의 생성과 소비를 나타낸다.

4.1.3 공정성 보장 정책에 따른 크레디트 및 서버요금 관리의 구현

각 스트림이 보유한 크레디트의 총량은 시간이 지남에 따라 일정하게 유지되어야 한다. 이를 위하여 단위 시간당 소모되는 크레디트의 총량과 단위 시간당 생성되는 크레디트의 총량을 동일한 값으로 유지하도록 크레디트 및 서버요금을 관리한다.

연산이 수행되는 서버에 기반하여 해당 시스템의 총 수행능력인 크레디트 증가량을 결정하고 이를 각 스트림에 가중치에 따라 분배하는 방식을 취한다. 해당 관리 기법을 위한 수식에 사용된 표기법은 표 4에 열거되어 있으며, 각 수식 및 관련된 설명은 아래와 같다.

1) 시스템 단위의 공정성 보장 및 고정적 요금 부여

해당 공정성 보장 방식의 경우, 모든 서버가 동일한 상수 C값의 가격을 보유하고, 수행중인 서버의 가격을 모두 더한 값이 단위 시간당 소모되는 크레디트의 총량이 된다. 이때 해당 총 크레디트 소모량

표기법	설명
S_i	서버 i
ST_j	스트림 j
$P(S_i)$	서버 i의 가격
$W(ST_j)$	스트림 j의 가중치
$BL(S_i)$	서버 i에 백로그를 갖고 있는 스트림의 집합
B	해당 단위 시간에 연산이 수행되고 있는 서버의 집합
C	상수

표 4 크레디트 및 서버요금 관리 시 사용된 표기법

$$P(S_i) = C$$

$$C_{inc}(ST_j) = \sum_{S_i \in B} P(S_i) \times \frac{W(ST_j)}{\sum_{\forall k} W(ST_k)}$$

수식 5 시스템 단위의 공정성 보장 및 고정적 요금 부여시 서버가격 및 크레디트 증가량의 결정

을 시스템에 백로그를 가진 스트림이 가중치에 따라 분배받은 값만큼 크레디트를 증가시킴으로써 크레디트의 총량이 유지됨과 동시에 시스템 단위의 공정성을 보장받게 된다. 이 동작은 수식 5와 같다.

2) 시스템 단위의 공정성 보장 및 가변적 요금

해당 공정성 보장 방식의 경우, 특정 서버는 해당 서버에 백로그를 갖는 스트림의 가중치의 총합에 비례하는 서버가격을 배정받는다. 이후 각 시스템에 백로그를 가진 스트림은 서버가격의 총량을 각 스트림의 가중치에 따라 분배받은 만큼 크레디트 증가량을 얻는다. 이 동작은 수식 6과 같다.

$$P(S_i) = C \times \sum_{ST_j \in BL(S_i)} W(ST_j)$$

$$C_{inc}(ST_j) = \sum_{S_i \in B} P(S_i) \times \frac{W(ST_j)}{\sum_{\forall k} W(ST_k)}$$

수식 6 시스템 단위의 공정성 보장 및 가변적 요금 부여시 서버가격 및 크레디트 증가량의 결정

$$P(S_i) = C$$

$$C_{inc}(ST_j) = \sum_{S_i \in B} \begin{cases} 0 & \text{if } ST_j \notin BL(S_i) \\ (P(S_i) \times \frac{W(ST_j)}{\sum_{k \in BL(S_i)} W(ST_k)}) & \text{if } ST_j \in BL(S_i) \end{cases}$$

수식 7 서버 단위의 공정성 보장 및 고정적 요금 부여시 서버가격 및 크레디트 증가량의 결정

3) 서버 단위의 공정성 보장 및 고정적 요금

해당 공정성 보장 방식의 경우 모든 서버에 대해 동일한 상수 C 값의 가격을 부여한다. 또한 각 스트림의 크레디트 증가량은 백로그를 보유한 서버에 대해 해당 서버에 백로그를 보유한 스트림 간의 가중치에 비례한 값만큼의 크레디트 증가량 산출해 모든 서버에 대해 합산하여 결정된다. 이 경우에도 모든 스트림의 크레디트 증가량의 합은 연산을 수행하고 있는 서버가격의 합과 동일하므로 시스템의 크레디트 총량은 일정하게 유지되게 된다. 이 동작은 수식 7과 같다.

4) 서버 단위의 공정성 보장 및 가변적 요금

해당 공정성 보장 방식의 경우 특정 서버는 해당 서버에 백로그를 갖는 스트림의 가중치의 총합에 비례하는 서버가격을 배정받는다. 이후 각 스트림의 크레디트 증가량은 서버 단위의 공정성을 보장하고

$$P(S_i) = C \times \sum_{ST_j \in BL(S_i)} W(ST_j)$$

$$C_{inc}(ST_j) = \sum_{S_i \in B} \begin{cases} 0 & \text{if } ST_j \notin BL(S_i) \\ (P(S_i) \times \frac{W(ST_j)}{\sum_{k \in BL(S_i)} W(ST_k)}) & \text{if } ST_j \in BL(S_i) \end{cases}$$

수식 8 서버 단위의 공정성 보장 및 가변적 요금 부여시 서버가격 및 크레디트 증가량의 결정

고정적 요금을 부여하는 방식과 동일한 접근 방식으로 산출한다. 이 때 총 크레디트 증가량과 총 서버가격의 합은 동일하게 유지된다. 이 동작은 수식 8과 같다.

제 5 장 공정대기열 스케줄링 기반 플래시 메모리 제어 기법

5.1 공정대기열 스케줄링 기반 플래시 메모리 제어 기법

공정대기열 스케줄링 기법은 네트워크의 패킷 처리 시스템에서 서버의 서비스율을 공정하게 분배함으로써 특정 스트림의 응답시간과 대역폭을 일정 수준까지 보장하는데 활용 되어 왔다. 이러한 방식은 호스트에서 유발된 플래시 메모리 요청과 플래시 메모리 저장장치에서 유발된 플래시 메모리 요청이 플래시 메모리 시스템의 서버인 버스와 칩에서 경쟁함으로 인해 발생하는 호스트 요청의 응답성 및 처리율을 보장하는 데에도 유효할 수 있다.

그러나 플래시 메모리 시스템이 갖는 다음과 같은 특징으로 인해 선행연구이외에 추가적인 연구가 필요하다.

5.1.1 플래시 메모리 시스템의 특징

플래시 메모리 시스템은 다음과 같은 특징을 지닌다.

1) 다수의 서버를 지닌 다중서버 환경

플래시 메모리 시스템은 데이터 전송 단위인 버스와 플래시 연산을 수행 단위인 플래시 메모리 칩으로 구성되어 있다. 시스템의

성능 및 용량의 요구사항에 따라 해당 버스의 개수와 칩의 개수는 다양하게 변화할 수 있으며, 해당 개수가 늘어날수록 시스템에서 제공하는 병렬성은 증가한다.

2) 공유서버가 존재하는 불균형적(Asymmetric) 구조

플래시 메모리 시스템은 하나의 버스를 다수의 플래시 메모리 칩이 공유하는 구조를 갖는다. 따라서 버스의 데이터 전송 속도와 플래시 메모리 칩의 연산 속도, 그리고 하나의 버스를 공유하는 칩의 개수에 따라 버스에 부과되는 부하가 달라질 수 있다.

3) 목표서버가 지정된 다중서버 환경으로 인한 서버부하의 역동성

플래시 메모리 시스템의 요청은 서비스를 받을 목표 버스 및 칩이 지정 되어 있으며, 따라서 특정 시점에 각 버스와 칩에서 경쟁하는 스트림의 집합과 각 스트림이 백로그를 갖는 버스와 칩의 집합이 시간에 따라 다양하게 변화할 수 있으며, 동시에 불균형적인 경쟁 상황이 발생할 수 있다.

4) 가변적인 서비스 시간을 갖는 비선점(Non-preemptive) 요청

플래시 메모리 연산인 읽기 연산, 쓰기 연산, 지우기 연산의 지연 시간의 분포는 플래시 메모리의 비트를 저장하는 방식에 따라 다를 수 있으며, 같은 플래시 메모리 주소를 갖는 곳에서 동작하는 연산이라도 수행 시 마다 변화할 수 있다. 또한 플래시 메모리 칩 내부의 한정된 자원을 효과적으로 활용하기 위하여 한 번 수행이 시작된 서버는 해당 수행이 종료될 때까지 다른 연산을 시작하지 않는 비선점성을 지닌다.

5) 사용 서버의 선행성 제약

플래시 메모리 연산은 연산의 종류에 따라 활용하는 서버의 종

류가 다를 뿐 아니라 서버의 활용 순서에도 제약이 따른다.

5.1.2 공정대기열 스케줄링 기반 플래시 메모리 제어 기법의 요구사항

위와 같은 특성에 고려 할 때, 공정성을 보장하는 플래시 메모리 제어기법이 제공해야 하는 요구사항은 다음과 같다.

1) 목표서버가 지정된 불균형적 구조를 갖는 다수의 서버에서 서비스 받는 다수의 스트림에 대해 공정성을 보장할 수 있어야 한다. 이때 서버의 개수가 매우 많을 경우에도 실제 시스템 관점에서 공정성 보장이 가능해야 한다.

2) 가변적인 서비스 시간을 갖는 플래시 연산에 대응하여 실시간으로 서비스율의 공정한 분배가 가능해야 한다.

3) 사용 서버의 선행성을 고려할 때에도 특정 서버가 유휴상태에 빠지지 않음과 동시에 각 스트림이 공정한 서비스를 받을 수 있도록 해야 한다.

위와 같은 요구사항을 고려할 때에 플래시 메모리 제어 기법은 다음과 같은 특성을 갖는 것이 바람직하다.

1) 간결성

다수의 플래시 메모리 서비스에 대해 부과되는 서비스가 역동적으로 변화함에 따라 플래시 메모리 서비스에 공정성을 부여하기 위해서는 공정성을 관리하는 기법의 복잡도가 낮아야 한다. 이는 자원이 한정되어 있는 플래시 메모리 저장장치에서의 동작을 고려할 때에도 필수적인 특성이라 볼 수 있다.

2) 실시간성

플래시 메모리 연산의 가변적인 서비스 시간을 실시간으로 반영할 수 있도록 할 때에 더 높은 수준의 공정성을 보장할 수 있다. 플래시 메모리 서비스 시간을 미리 예측하여 동작하는 것도 가능할 수 있다. 그러나 일반적으로 사용되는 MLC 기법에서는 각 페이지의 레벨에 따라 수행시간이 변화할 수 있으며, 높은 레벨에 해당할수록 수행시간의 평균으로부터 편차가 증가하는 경향성을 보인다. 또한 위와 같이 서비스 시간을 예측하여 동작하는 경우 플래시 연산이 대상으로 하는 페이지의 레벨을 추출하고 또 해당 레벨의 평균 서비스 시간을 보유하는 등의 추가적인 정보관리가 필수적이며 이는 간결성을 저해하는 원인이 될 수 있다.

3) 효율성

플래시 메모리 연산의 선행성 제약으로 인해 특정 서버가 유휴상태에 빠질 가능성이 있으며 이를 최소화 하며 동작해야 함으로써 효율성을 보장해야 한다. 예를 들어 읽기 연산의 경우 데이터를 플래시 읽기 연산을 통해 페이지 버퍼에 준비를 해둔 후에야 버스를 통한 데이터전송을 수행할 수 있으며, 이 때 버스에서의 경쟁으로 인해 데이터전송이 수행되지 못할 경우 해당 읽기 연산이 수행된 플래시 메모리 칩은 다른 연산을 수행하지 못하게 되는 상황이 가능하다.

5.1.3 공정대기열 스케줄링 기반 플래시 메모리 제어 계층

본 연구에서는 공정성을 보장하는 플래시 메모리 제어 계층을 소프트웨어로 구현한다. 공정성을 보장해야 하는 스트림의 개수는 이론적으로 무한히 많이 존재할 수 있으며, 따라서 해당 스트림에 대해 공

정성을 보장하기 위해서는 충분한 자원과 유연성이 보장되어야하기 때문이다. 또한 이를 통해 고속으로 플래시 메모리 시스템의 자원을 관리해야 하는 하드웨어 모듈을 간결하게 유지함으로써 플래시 메모리 제어기의 동작특성과 구현 복잡도를 줄일 수 있는 장점이 있다.

소프트웨어 계층에서 플래시 메모리 시스템의 스트림에 대한 공정성을 간결하고 효율적으로, 그리고 실시간으로 관리하기 위해서는 플래시 메모리 시스템의 자원인 버스와 칩의 상태를 확인하는 인터페이스가 필수적으로 지원되어야 한다.

5.2 공정대기열 플래시 제어 계층의 구현

5.2.1 공정대기열 플래시 제어 계층의 구조

공정대기열 플래시 제어 계층은 4장에서 제안한 목표서버가 지정된 다중서버 환경에서의 공정성 보장 기법을 기반으로 동작한다. 해당 계층은 공정성 보장 방식에 따라 크레디트 및 서버요금을 관리하는 크레디트 관리 모듈과 특정 버스 및 칩에 수행할 요청을 선정하는 수락제어 모듈을 포함하는 공정성 관리모듈, 각 스트림의 요청에 대해 목표 버스 및 칩별로 요청을 관리하는 요청 대기열, 외부의 요청을 수용함과 동시에 사용자의 환경설정을 반영하는 공정대기열 제어계층 인터페이스, 플래시 메모리 제어기와의 연동을 지원하는 플래시 메모리 시스템 인터페이스로 구성된다. 해당 구조는 그림 18과 같다.

각 스트림별로 유지되는 버스 및 칩에 대한 별도의 요청대기열은 해당 스트림의 특정 서버에 대한 백로그 존재 유무를 확인하는 시간을 단축시키며, 공정성 관리 정책에 따라 수행이 허가된 요청이 빠른 시간 안에 무순서 플래시 제어기로 전달될 수 있도록 한다. 이와 같은

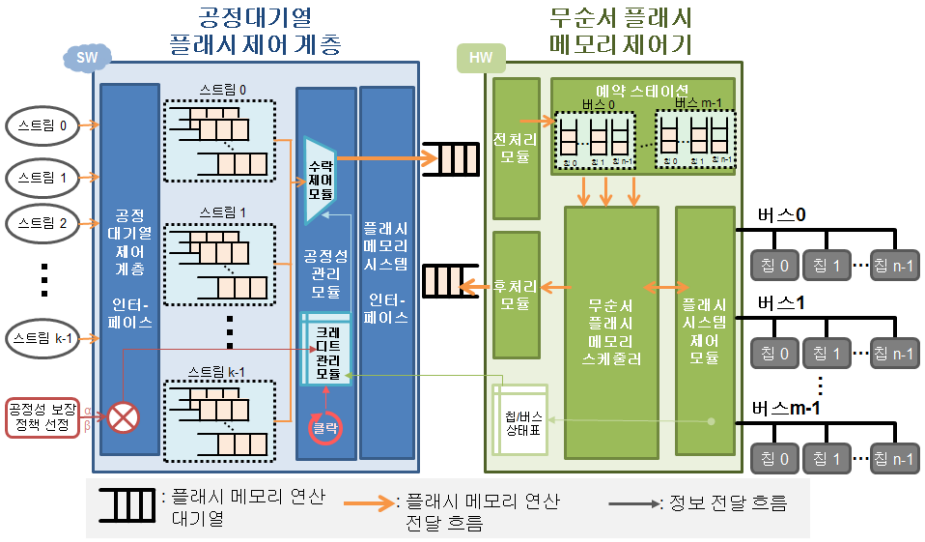

그림 18 공정대기열 플래시 제어 계층의 구조

수락제어 관리를 통해 전달된 요청은 목표 버스와 칩의 조합에 따라 무순서 플래시 메모리 제어기 내부에 목표 버스와 칩의 조합별로 유지되는 요청대기열에 전달되게 되며 해당 요청대기열에는 최대 1개의 요청이 대기하게 된다. 따라서 무순서 플래시 메모리 제어기에서 서비스 받고 있는 요청까지 고려하여 특정 버스와 칩을 서비스 대상으로 하는 요청이 최대 2개까지 무순서 플래시 제어기 내부에 존재하게 된다.

이때 공정대기열 플래시 제어 계층이 버스와 칩의 상태정보를 확인할 수 있도록 플래시 메모리 제어 계층의 인터페이스가 확장되어야 한다. 해당 인터페이스는 메모리 맵 입출력(Memory mapped I/O)의 형태를 통해 버스와 칩의 상태를 표현하는 레지스터에 접근하는 방식으로 구현이 가능하다.

5.2.2 공정대기열 플래시 제어 계층의 공정성 관리 기법의 구현

각 스트림별로 별도로 유지되는 크레디트는 크레디트 관리 모듈 내에 존재한다. 해당 크레디트의 변화는 시스템이 유지하는 클락의 속도에 비례하는 특정 시간 단위가 지남에 따라 실시간으로 변화된다. 공정성 관리 모듈은 수락제어 정보를 유지함으로써 특정 버스와 특정 칩에서 현재 주행되고 있는 요청이 어떤 스트림의 요청인지를 알 수 있다. 동시에 칩/버스 상태표를 통해 해당 요청의 서버 사용여부를 확인할 수 있다. 크레디트 관리 모듈은 공정성 보장 정책 및 요청대기열의 백로그 정보를 조합하여 서버의 요금을 산출하며, 해당 시간 동안 서버를 활용한 스트림의 크레디트를 감소시킨다. 또한 마찬가지 방식으로 요청대기열의 백로그 정보를 조합하여 특성 시점에 스트림의 크레디트 증가량을 산출하여 크레디트를 증가시킨다. 이과 같은 크레디트 관리는 서버의 종류에 구분 없이 진행한다.

수락제어 모듈의 경우, 칩/버스 상태표를 통해 특정 서버의 요청 종료 여부를 확인한다. 해당 요청이 종료되어 서버의 활용이 가능해지는 순간 크레디트 관리 모듈에서 유지하는 스트림별 크레디트의 정보와 요청대기열의 백로그 여부를 조합하여 해당 서버를 활용할 요청을 무순서 플래시 메모리 제어기에 전달한다.

이 때 하나의 칩만을 활용하는 지우기 연산의 경우, 해당 칩이 유휴상태에 들어가는 순간 전달해도 무방하다. 그러나 버스와 칩을 모두 사용하는 읽기 연산, 쓰기 연산의 경우 추가적인 수락제어 정책이 필요하다.

1) 쓰기 연산의 관리

쓰기 연산의 경우 버스와 칩을 동시에 활용하는 데이터 전송 작업 이후에 칩을 활용하는 지연시간동안 버스활용 없이 칩만을 활용한다. 따라서 쓰기 연산의 수락제어는 버스와 칩이 모두 수행가능한 상태일 때를 기준으로 관리한다.

2) 읽기 연산의 관리

읽기 연산의 경우 칩만을 활용하는 읽기 지연시간 이후에 버스와 칩을 동시에 활용하는 데이터 전송 작업이 수행된다. 따라서 특정 시점에 칩이 수행가능한 상태가 되어 해당 읽기 작업을 수행했을 지라도 읽기 작업이 완료된 이후의 데이터 전송 작업시 버스의 경쟁에서 더 낮은 크레디트를 갖는 상황이 발생할 수 있다.

해당 소프트웨어 계층은 무순서 플래시 메모리의 버스 및 칩의 스케줄링 방식을 활용하기 때문에 실시간으로 버스에서의 경쟁을 관리할 수는 없다. 따라서 수락제어 시점에서 차후에 일어날 버스의 경쟁을 미리 고려하여 해당 요청의 전달 여부를 결정해야 한다.

이와 같이 활용하는 서버의 종류와 그 순서가 다른 상황에 대응하기 위하여 읽기 및 쓰기 연산의 경우 버스와 칩 모두에 대해 항상 백로그를 갖는 연산으로 취급한다. 이를 통해 읽기 연산이 수락제어를 받는 시점에 버스와 칩 모두에 대해 우선적인 수행을 보장받도록 하며, 그 이후의 버스에서의 경쟁은 무순서 플래시 메모리 제어기의 정책을 따름으로써 시스템 전체의 수행효율을 보장하도록 한다.

제 6 장 실험 및 평가

6.1 실험 환경

실험환경은 본 연구진에서 개발한 고해상 실시간 시뮬레이터를 기반으로 진행한다. 플래시 메모리의 구성은 다수의 서버에 대응하는 환경을 반영하도록 하며, 지연시간의 특성은 2-bit MLC를 선정하여 플래시 메모리의 다양한 지연시간 분포를 반영하도록 한다. 또한 스트림의 생성은 실험의 목적에 따라 다양하게 설정할 수 있도록 한다.

실험에서 사용된 고해상 실시간 시뮬레이터의 플래시 메모리 환경 설정은 표 5와 같다.

		설정 값
버스 개수		8
버스 당 칩 개수		8
칩 당 블록 개수		256
블록 당 페이지 개수		256
페이지 크기		4314bytes (데이터영역 : 4KB, 여유영역 : 218bytes)
버스 데이터 전송률		33MB/s
읽기 연산 지연시간	레벨 0	20us
	레벨 1	50us
쓰기 연산 지연시간	레벨 0	200us
	레벨 1	800us
지우기 연산 지연시간		2000us

표 5 고해상 실시간 시뮬레이터의 환경설정 변수

6.2 공정성 보장 여부 확인 실험

6.2.1 균등한 경쟁에서의 공정성 보장 확인

목표서버가 지정된 다중서버 환경에서 모든 스트림이 모든 서버에 대해 백로그를 지닌 경우, 해당 환경은 목표서버와 상관없이 모든 서버에서 동작이 가능한, 일반적인 다중서버 환경에서의 동작과 동일한 동작 특성을 보인다. 따라서 해당 서비스는 공정성 보장 정책의 종류와 상관없이 동일한 동작을 보인다. 모든 공정성 보장 정책이 동일한 크레디트 증가양과 동일한 서버요금을 산출되는 것으로 해당 동작의 동일성을 확인할 수 있다.

하나의 버스만을 가정하여 8개의 스트림이 8개의 칩이 존재하는 버스에서 경쟁을 하는 경우, 20초 동안의 수행에 대한 실험결과는 다음과 같다. 이때 각 스트림은 동일한 가중치를 갖는다. 그리고 각 스트림은 임의의 플래시 메모리 칩, 블록, 페이지에 대해 임의의 읽기 및 쓰기 연산을 생성한다. 해당 실험 결과는 표 6, 그림 19와 같다.

먼저 전체 서비스 시간과 버스에서의 서비스 시간은 각 스트림별로

	버스	칩 0	칩 1	칩 2	칩 3	칩 4	칩 5	칩 6	칩 7	총계
스트림 0	2453.4	1514.3	1442.6	1576.9	1499.3	1470.1	1483.7	1463.9	1473.6	14377.8
스트림 1	2453.5	1496.7	1508.5	1437.6	1452.6	1546.0	1535.5	1507.8	1442.3	14380.4
스트림 2	2453.5	1502.7	1538.5	1506.3	1500.4	1430.9	1476.2	1450.6	1532.1	14391.3
스트림 3	2453.4	1495.4	1488.3	1474.1	1482.2	1499.9	1464.3	1500.8	1496.5	14354.9
스트림 4	2453.4	1515.6	1534.8	1439.7	1489.7	1478.8	1444.7	1524.4	1483.0	14364.1
스트림 5	2453.5	1452.5	1446.8	1484.2	1511.8	1527.8	1519.9	1486.3	1519.1	14401.8
스트림 6	2453.4	1490.2	1479.8	1538.7	1474.1	1514.7	1538.7	1435.1	1466.2	14390.9
스트림 7	2453.5	1488.4	1551.4	1467.1	1524.9	1487.3	1460.5	1500.7	1457.1	14390.9
총계	19627.6	11955.9	11990.8	11924.5	11935.0	11955.5	11923.4	11869.5	11869.8	115052.1

표 6 스트림 별 버스 및 칩에 대한 서비스 시간 표 (단위 : ms)

유사한 서비스비율을 보인다. 칩에서의 경쟁의 경우 스트림에 따른 서비스 시간에 다소 차이가 있으나, 전체적인 시스템에서의 서비스양은 일정함을 확인 할 수 있다. 이때 칩에서와는 달리 버스에서 경쟁이 공정하게 반영되는데 이는 버스가 공유자원으로서 높은 비중의 처리율을 보이기 때문이다. 20초 동안의 실험시간 동안 버스가 서비스된 시간은 총 19.63초로서 이는 98.14%의 사용률을 나타낸다. 반면 각 칩의 경우 평균적으로 59.6%의 사용률을 보였다. 따라서 칩에서 서비스의 불균형으로 보이는 서비스 시간의 차이는 사실 버스에서의 높은 경쟁으로 인해 각 칩이 서비스를 받지 못함으로서 발생한 현상임을 알 수 있다. 이와 같은 동작은 DRFQ과 같은 방식에서 의도하는, 더 높은 상대적 서비스 비중을 갖는 자원을 기준으로 공정성을 보장하는 것과 유사하다고 볼 수 있다.

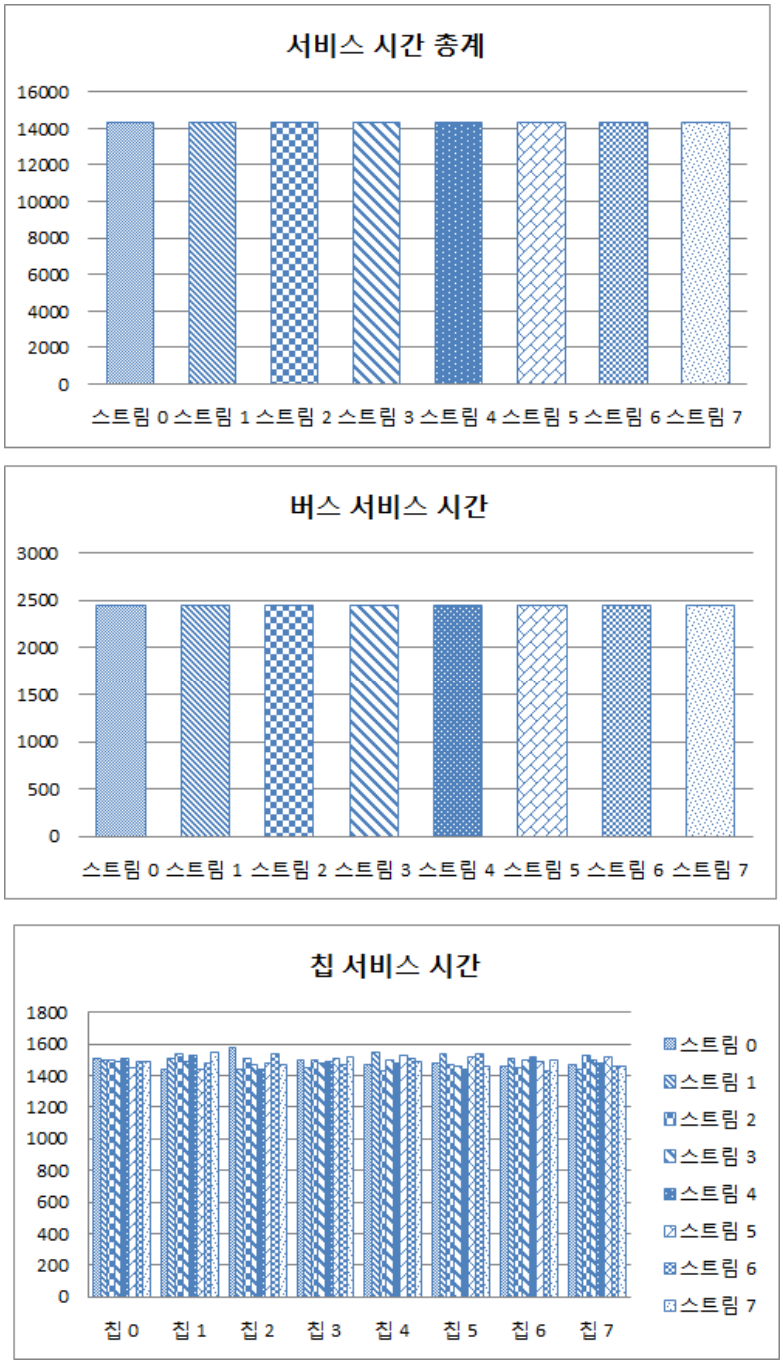

그림 19 균등한 경쟁에서의 공정성 보장 실험 결과

6.2.2 서로 다른 경쟁에서의 공정성 보장 확인

목표서버가 지정된 다중서버 환경에서 각 스트림이 서버에 대해 백로그를 지닌 분포가 다른 경우, 각 공정성 보장 정책에 따라 서로 다른 서비스 제공 패턴을 보인다.

각 공정성 보장 정책의 특징을 직관적으로 이해하기 위하여 먼저 2개의 스트림이 2개의 칩에 대해 서로 다른 경쟁을 갖는 경우를 분석한 후 8개의 스트림이 8개의 칩에 대해 서로 다른 경쟁을 갖는 경우와 4개의 스트림이 8개의 칩에 대해 서로 다른 경쟁을 갖는 경우를 살펴보고자 한다. 이때, 공유되는 자원에서 발생하는 효과를 제거하기 위하여 버스에서의 동작 없이 칩에서의 동작만을 필요로 하는 경우를 가정하였다. 또한 논의의 단순화를 위하여 각 스트림이 동일한 가중치를 갖는 경우를 가정하였다.

실험 1) 칩 2개, 스트림 2개의 환경에서 서로 다른 경쟁

*스트림 0*는 *칩 0*에, *스트림 1*은 *칩 0*과 *칩 1*에 백로그를 갖는 경우, 각 공정성 보장 정책에 따른 서비스 시간 분배 결과는 그림 20과 같다.

i) 시스템 단위의 백로그를 갖고 고정적 요금을 부여한 경우

해당 공정성 보장 정책의 경우에는 시스템 전체의 서비스 시간이 동일하게 분배됨을 확인할 수 있다. 다만 이 경우 *칩 1*에 백로그를 갖지 않는 *스트림 0*가 *칩 0*을 독점적으로 점유하고 *스트림 1*이 *칩 1*을 독점적으로 점유함으로써 시스템 전체의 서비스 시간이 균형을 잡게 된다. 이에 칩 0에서 처리되어야 하는 스트림 1의 요청이 기아현상을 경험하게 된다. 따라서 해당 공정성 보장 정책은 적합한 부하 균등화 정책과 동시에 지원되어야 한다.

그림 20 서로 다른 경쟁에서의 공정성 보장 실험 결과 (1)

ii) 서버 단위의 백로그를 갖고 고정적 요금을 부여한 경우

해당 공정성 보장 정책의 경우에는 경쟁이 일어나는 칩을 단위로 서비스 시간이 균등하게 분배됨을 확인할 수 있다. 이와 같은 동작을 통해 실제 전체 플래시 메모리 기반 저장장치가 특정 스트림에 부여된 경우에 활용되는 병렬성을 감안한 공정한 서비스 분배가 가능하다.

iii) 시스템 단위의 백로그를 갖고 가변적 요금을 부여한 경우

해당 공정성 보장 정책의 경우에는 경쟁이 발생하는 *칩 0*에서 다른 칩에서 서비스를 받는 *스트림 1*이 보다 불리한 경쟁 상황을 맞이하게 된다. 이는 같은 경쟁상황에서 각 자원에 대한 독점적인 점유가 발생하는 시

스템 단위의 백로그를 갖고 고정적 요금을 부여한 경우와 달리 서로 다른 서버의 서비스 가치를 부여함에 따른 결과라고 볼 수 있다.

iv) 서버 단위의 백로그를 갖고 가변적 요금을 부여한 경우

해당 공정성 보장 정책의 경우에는 서버 단위의 백로그를 갖고 고정적 요금을 부여한 경우와 동일한 동작 특성을 보인다. 이는 특정 스트림의 크레디트가 생성되는 양 만큼 해당 서버에서 크레디트가 소모되는 특성으로 인한 서버 단위의 경쟁이 보장되기 때문이다.

실험 2) 칩 8개, 스트림 8개의 환경에서 서로 다른 경쟁

각 스트림이 자신의 스트림 번호 이하의 칩에 백로그를 갖는 경우 (*스트림 0*는 *칩 0*에, *스트림 1*은 *칩 0*과 *칩 1*에, *스트림 2*는 *칩 0*과 *칩 1*과 *칩 2*에 …….) 각 공정성 보장 정책에 따른 서비스 시간 분배 결과는 그림 21과 같다. 각 공정성 보장 정책의 특성에 따라 서버 단위의 백로그를 기준으로 공정성을 보장하는 경우 각 서버 내부에서 공정한 서비스 시간의 분배가 이루어졌다. 시스템 단위의 백로그를 기준으로 공정성을 보장하는 경우, 고정적 요금을 부여할 경우 모든 스트림의 시스템에서 사용한 서비스 타임의 합이 동일하게 관리됨을 볼 수 있다. 또한 가변적 요금을 부여할 경우 보다 적은 수의 서버에 서비스를 요청하는 스트림이 해당 서버에서의 경쟁에서 더 많은 서비스 시간을 분배받게 됨을 볼 수 있다.

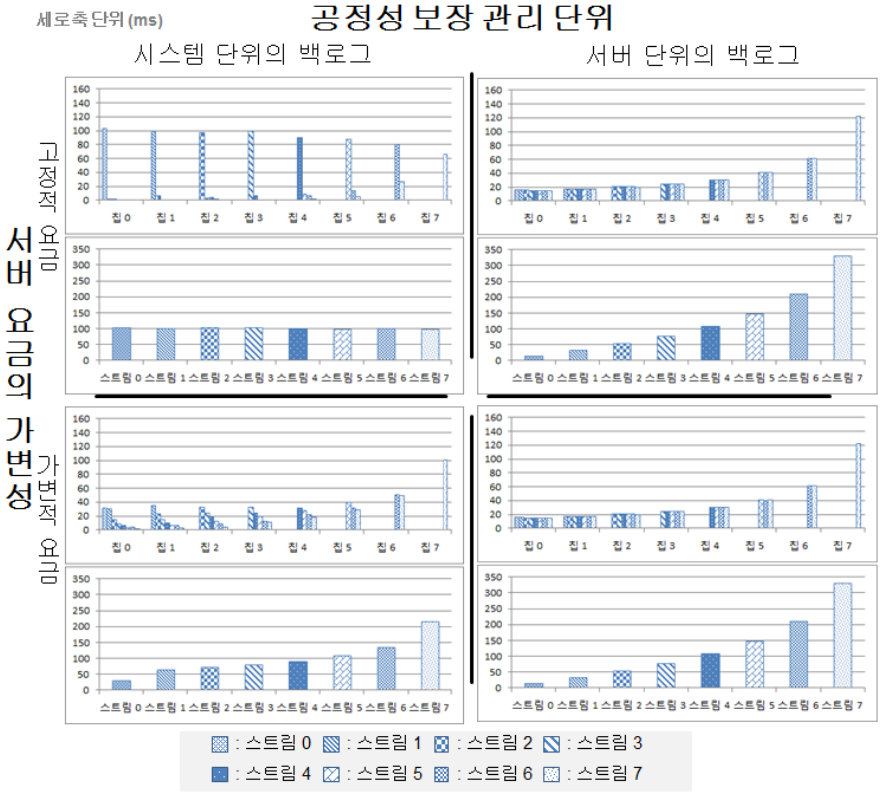

그림 21 서로 다른 경쟁에서의 공정성 보장 실험 결과 (2)

실험 3) 칩 8개, 스트림 4개의 환경에서 서로 다른 경쟁

표 7과 같은 설정의 백로그를 갖는 경우 서비스 시간의 분배 결과는 그림 22와 같으며 전체 서버의 수보다 더 작은 수의 스트림이 경쟁할 때에도 실험 1, 실험 2에서 설명한 것과 동일한 동작 특성을 보임을 확인할 수 있다. 단, 가변적 요금을 부여하고 시스템 단위로 공

	스트림 0	*스트림 1*	*스트림 2*	*스트림 3*
백로그 된 서버	*칩 0, 칩 1*	*칩 1, 칩 2, 칩 3*	*칩 2, 칩 3, 칩 4*	*칩 4, 칩 5, 칩 6, 칩 7*

표 7 서로 다른 경쟁 실험 3의 스트림 별 백로그된 서버 설정

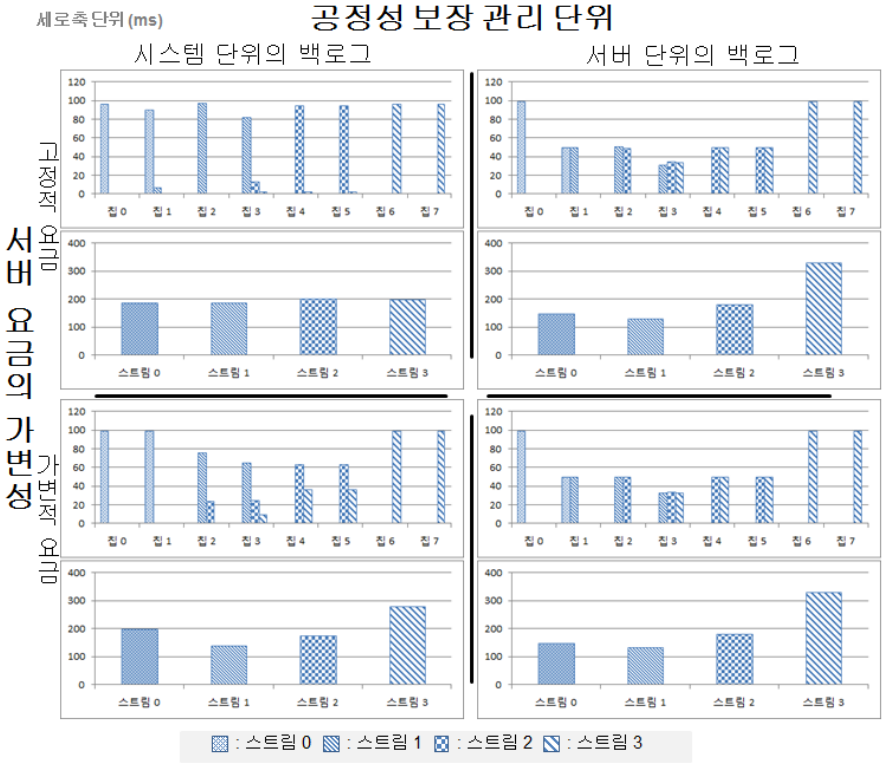

그림 22 서로 다른 경쟁에서의 공정성 보장 실험 결과 (3)

정성을 보장하는 경우 *칩 1*에 대한 *스트림 1*의 요청에서 기아현상이 발생하게 됨을 볼 수 있다. 이 현상은 경쟁이 상대적으로 적은 *칩 0*와 *칩 1*에서 서비스를 받는 *스트림 0*가 *칩 0*와 *칩 1*을 모두 점유하여도 제공받는 서비스 가치의 합이 다른 스트림보다 더 적게 유지되기 때문에 발생하였음을 확인하였다. 이에 시스템 단위의 공정성을 보장하는 경우 적합한 부하균등화 기법이 필요함을 다시 한 번 확인하였다.

6.2.3 서로 다른 가중치를 갖는 경우 공정성 보장 확인

실험 1) 8개의 칩에 8개의 스트림이 경쟁하는 경우

모든 스트림이 모든 칩에 백로그를 갖고 스트림 번호에 따라 가중치가 선형적으로 증가하는 경우(스트림 0 : 1, 스트림 1 : 2, 스트림 2 : 3 ... 스트림 7 : 8), 공정성 보장 정책에 따른 서비스 시간의 분배 결과는 그림 23과 같다.

모든 칩에 모든 스트림이 백로그를 갖는 경우, 각 서버의 관점에서 그리고 전체 시스템의 관점에서 가중치에 비례하는 서비스 시간이 분배되고 있음을 확인할 수 있다.

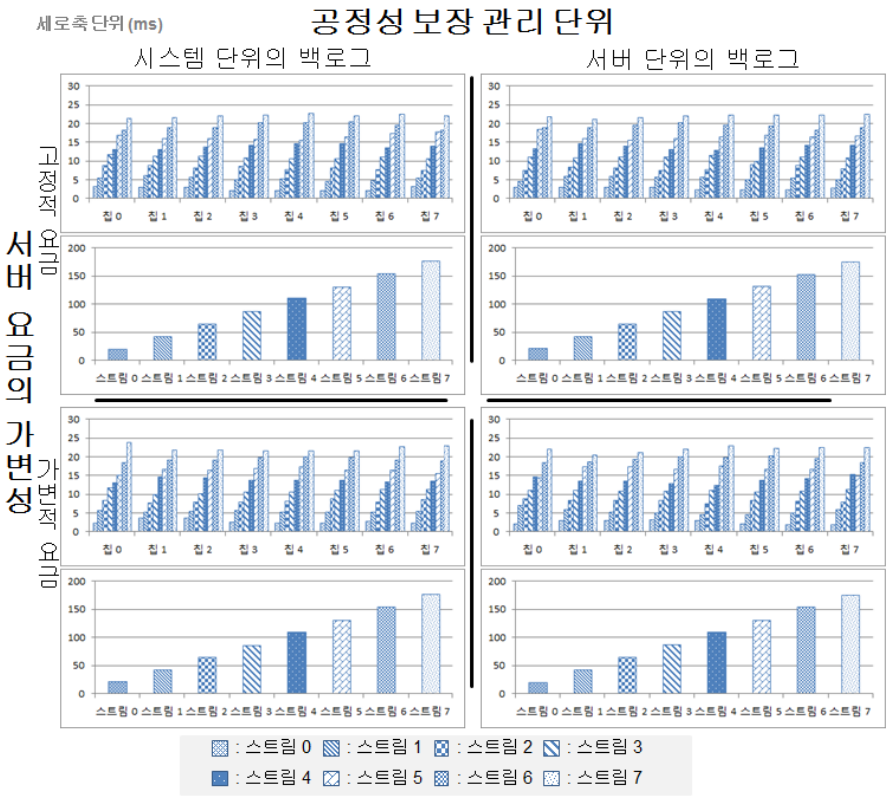

그림 23 서로 다른 가중치를 갖는 경우 공정성 보장 실험 결과 (1)

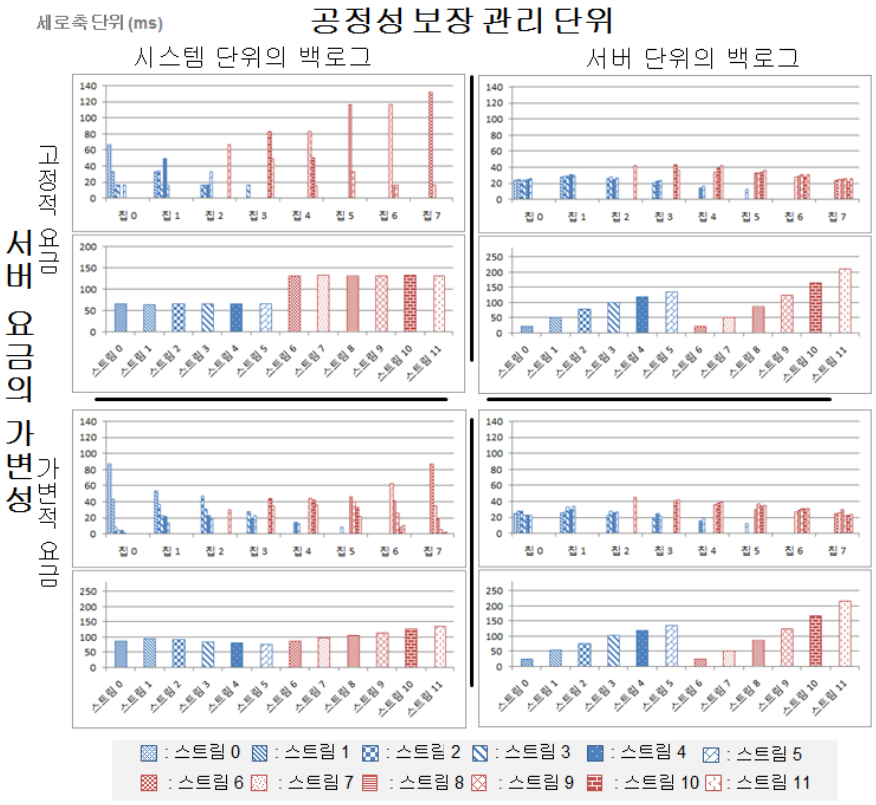

그림 24 서로 다른 가중치를 갖는 경우 공정성 보장 실험 결과 (2)

실험 2) 8개의 칩에 12개의 스트림이 경쟁하는 경우

스트림의 백로그된 서버와 각 스트림의 가중치의 설정이 표 8과 같을 때 공정성 보장 방식에 따른 서비스 시간 분배 결과는 그림 24와 같다.

이를 통해 서버보다 더 많은 수의 스트림이 서로 다른 가중치를 갖고 각 서버에서의 경쟁이 다를 경우에도 공정성 보장 방식이 올바르게 동작함을 볼 수 있다. 먼저 시스템 단위로 공정성을 보장하고 고정적 요금을 부여한 경우 가중치에 비례하는 총 시스템 서비스 시간의 분배가 이루어진다. 또한 시스템 단위의 공정성을 보장하고 가변

스트림 (가중치 1)	*스트림 0*	*스트림 1*	*스트림 2*	*스트림 3*	*스트림 4*	*스트림 5*
백로그 된 서버	*칩 0*	*칩 0, 칩 1*	*칩 0, 칩 1, 칩 2*	*칩 0, 칩 1, 칩 2, 칩 3*	*칩 0, 칩 1, 칩 2, 칩 3, 칩 4*	*칩 0, 칩 1, 칩 2, 칩 3, 칩 4, 칩 5*
스트림 (가중치 2)	*스트림 6*	*스트림 7*	*스트림 8*	*스트림 9*	*스트림 10*	*스트림 11*
백로그 된 서버	*칩 7*	*칩 6, 칩 7*	*칩 5, 칩 6, 칩 7*	*칩 4, 칩 5, 칩 6, 칩 7*	*칩 3, 칩 4, 칩 5, 칩 6, 칩 7*	*칩 2, 칩 3, 칩 4, 칩 5, 칩 6,, 칩 7*

표 8 서로 다른 가중치 실험 2의 스트림 별 백로그된 서버 설정

적 요금을 부여한 경우 각 스트림이 경쟁에 참여하는 서버의 개수에 반비례하는 서비스 시간 분배가 발생하며, 그 경향성이 가중치에 비례하고 있음을 확인할 수 있다.

또한 서버 단위의 백로그를 기준으로 공정성을 보장하는 경우 특정 서버에서 경쟁하는 스트림 사이에서 가중치에 비례하는 서비스 시간 분배가 이루어짐을 확인할 수 있다.

6.2.4 공유자원을 포함한 경우 공정성 보장 확인

플래시 메모리 시스템은 버스라는 공유자원가 존재한다는 점에서 기존의 다중서버 시스템과 차별성을 갖는다. 하나의 버스를 2개의 칩이 공유하는 시스템에서 *스트림 0*가 *칩 0*에, *스트림 1*이 *칩 0*과 *칩 1*에 버스를 활용하는 쓰기 및 읽기 연산을 발생시킬 경우 공정성 보장 방식에 따른 서비스 시간의 분배 결과는 그림 25와 같다.

이 때 시스템 단위의 공정성을 보장하고 고정적 요금을 부여하는 경우 예상과 동일한 시스템 총량의 합에서 공정한 서비스 분배가 발생함을 확인 할 수 있다. 또한 시스템 단위의 공정성을 보장하고 가변적 요금을 부여하는 경우, 기존 칩에서의 경쟁만을 고려했을 때에는 스트림 1에 의해 완전히 점유되었던 *칩 1*에 유휴시간이 발생하고

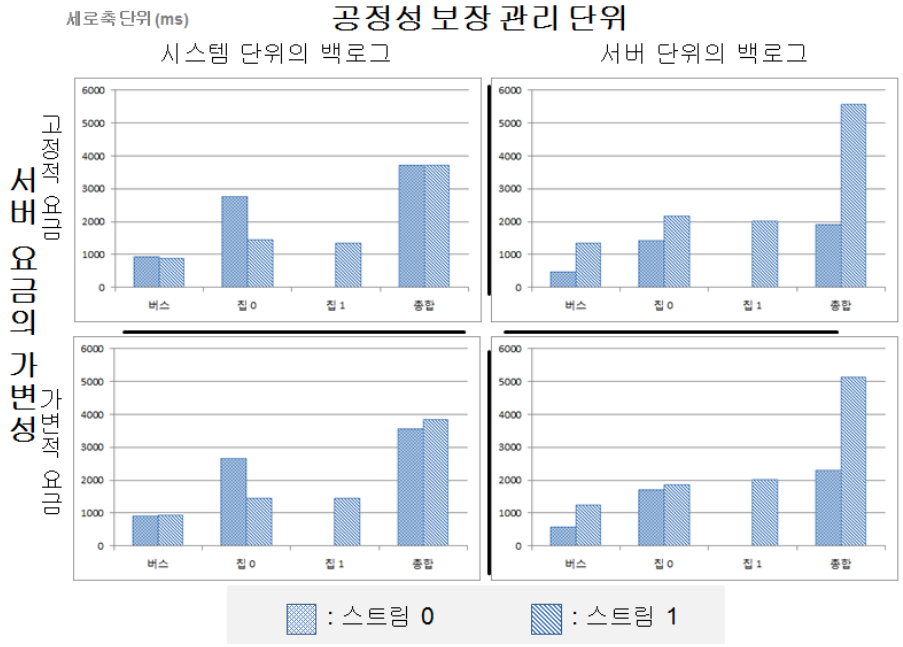

그림 25 공유자원을 포함한 경우 공정성 보장 실험 결과

있음을 확인할 수 있다. 이는 두 스트림이 동시에 공유하는 버스에서의 경쟁에서 *칩 1*에 부여되었던 크레디트가 소모되기 때문으로 생각된다.

서버 단위의 공정성을 보장하고 고정적 요금을 부여하는 경우, 각 서버에서 동등한 서비스 분배가 이뤄져야 하나, *칩 0*와 *칩 1* 모두에 백로그를 갖는 *스트림 1*이 버스와 칩 0에 대해 더 높은 점유율을 보이고 있음을 확인할 수 있다. 이는 *스트림 1*이 *칩 1*을 통해 독점적으로 받는 크레디트 증가량을 버스와 *칩 0*에서 사용하고 있기 때문으로 생각된다.

이와 같이 공유자원이 존재할 경우, 과점유 되고 있는 서버에서 발생한 크레디트가 공유자원에서 사용됨으로써 공정한 서비스 분배가 이루어지게 됨을 확인할 수 있다.

6.3 응답성 보장 여부 확인 실험

6.3.1 새로운 스트림의 경쟁 참여시 응답성 보장 여부 확인

공정성 보장을 통해 얻고자 하는 이득은 서비스 시간 분배를 통한 처리율의 공정한 분배와 함께, 이러한 처리율의 공정한 분배를 통한 응답성의 보장여부이다. 해당 응답성을 보장하는 접근 중, 기존 방식과 가장 큰 차이점은 스트림별로 서로 다른 대기열을 유지함으로써 다른 스트림이 생성한 다수의 요청이 플래시 메모리 컨트롤러의 요청 대기열에 존재하고 있는 경우에도 새롭게 경쟁에 참여한 스트림의 응답성이 보장된다는 것이다.

해당 특성을 확인하기 위하여 동일한 가중치를 갖는 2개의 스트림인 *스트림 0*와 *스트림 1*에 대해 다음과 같은 실험을 진행하였다.

*스트림 0*는 매 100us당 하나의 플래시 요청을 생성한다. *스트림 1*은 *스트림 0*가 2000개의 요청을 생성한 이후에 매 100us당 요청을 생성한다. 이와 같은 요청 생성 패턴을 기존의 플래시 메모리 시스템과 공정성 보장 방식을 적용한 시스템에 적용하였다. 각 경우에 요청이 발생한 시간에 따른 응답시간의 분포는 그림 26와 같다.

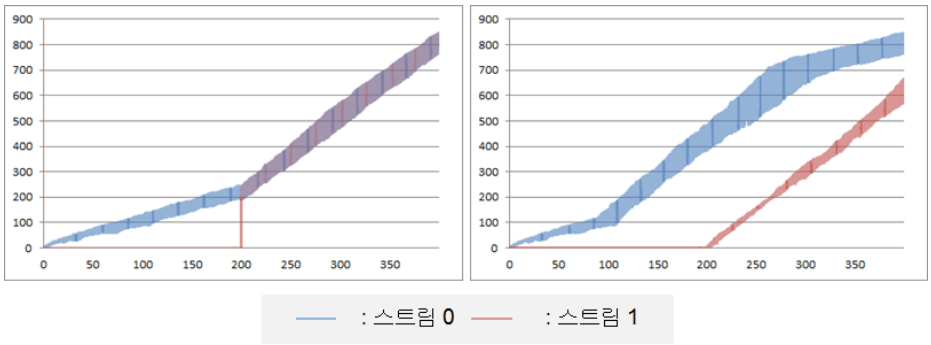

그림 26 요청 발생 시간에 따른 응답시간의 분포
(좌 : 기존 방식, 우 : 공정성 보장 방식)

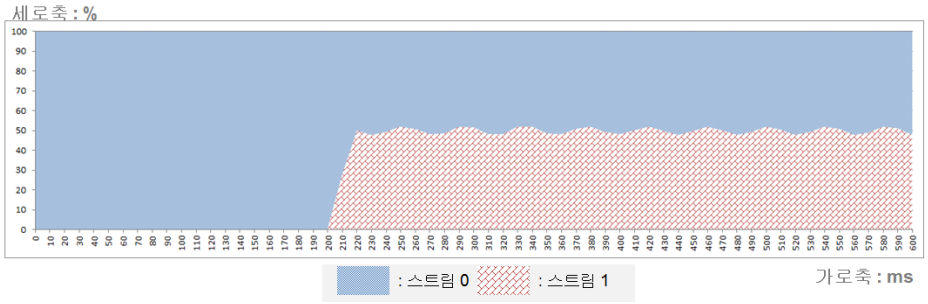

그림 27 시간에 따른 스트림별 시스템 서비스 시간의 분배

이때 기존 방식의 경우 새로운 요청이 요청되었을 때 선행적으로 발생한 다른 요청이 종료된 이후에 처리가 가능함으로 응답시간이 길게 발생하고 있음을 확일 할 수 있다. 반면 공정성 보장 방식의 경우 새로운 요청 스트림이 경쟁에 참여할 때에 공정성 처리 방식에 따라 곧바로 처리가 가능함을 확인 할 수 있다. 이때 공정성 처리 방식을 적용한 시스템에서 각 시간에 따른 스트림 별 서비스 시간의 분배는 그림 27과 같다.

따라서 플래시 메모리 내부 관리 기능에 의해 플래시 메모리 시스템이 점유되었을 때에 긴급한 호스트 요청이 발생한 경우 기존의 시스템에서는 해당 호스트 요청의 응답시간이 내부 관리 기능에 의해 먼저 발생한 플래시 메모리 요청에 의해 결정되지만, 공정성 보장 기법을 활용한 경우 기존에 발생한 요청과 상관없이 새롭게 경쟁에 참여한 스트림의 가중치에 의해 요청의 응답시간이 결정된다.

제 7 장 결론

플래시 메모리 저장장치는 호스트의 요청으로 인해 유발되는 플래시 메모리 연산과 내부관리로 인해 발생하는 플래시 메모리 연산이 경쟁함에 따라 호스트 요청의 응답성 및 처리율에 변동이 발생할 수 있다. 이러한 성능의 불안정성은 플래시 메모리 저장장치를 활용하는 여러 시스템의 성능저하로 직결될 수 있으며 해당 문제를 체계적으로 효과적으로 관리하기 위한 해결책이 필요하다.

이에 네트워크 패킷 전송 시스템에서 응답시간과 대역폭을 보장하기 위하여 사용된 공정대기열 알고리즘을 플래시 메모리 저장장치의 환경에 적용하여 응답시간 및 처리율의 안정성을 제공하는 방식을 제안하였다. 그러나 기존의 공정성 보장을 위한 기법들은 플래시 메모리 저장장치와 같이 목표서버가 지정된 다중서버 환경에 직접 적용하는 데 한계가 있었다.

먼저 단일 서버를 대상으로 하는 공정대기열 알고리즘인 WFQ, SFS, SCFQ, WF^2Q등은 단일 서버상의 공정성에 대해서는 높은 수준의 공정성을 보장할 수 있으나, 플래시 메모리 시스템과 같이 다수의 서버를 지닌 시스템에서의 공정성을 보장할 수 없다.

또한 다중 서버를 대상으로 하는 경우 부하 균등화를 통하여 단일 서버의 상황으로 근사화 하는 접근을 취하는, 링크 병합 공정대기열 방식 및 CFS와 같은 방식이 있었다. 그러나 해당 방식들이 다루는 환경과 달리 플래시 메모리 시스템은 목표 서버가 지정된 요청 특성을 갖기 때문에 부하 균등화가 불가능한 특성이 있어 새로운 접근 방식이 필요로 했다.

더 나아가 다중 서버를 대상으로 하며 목표 서버가 지정된 요청을 다루는 선행연구로서 인터넷 라우터에서 출력링크별로 공정성을 보장

하고자 하는 방식이 도입되었다. 그러나 해당 방식은 다수의 가상 시간을 관리함으로써 구현 복잡도가 높고 더 나아가 출력링크별로 입력링크에 대한 공정성을 독립적으로 보장하는 방식을 취함으로써 백로그의 불균형이 있을 경우 시스템 전체의 관점에서 입력링크에 대한 공정성을 보장할 수 없다는 단점을 지니고 있다. 이와 달리 본 연구에서 제안한 공정성 보장 정책은 서버 단위의 공정성과 시스템 단위의 공정성을 모두 고려하고 있으며, 두 가지 방식을 크레디트라는 단일 변수로 관리함으로써 그 구현 복잡도도 단순하다는 장점이 있다.

또한 네트워크 패킷 전송의 중간자 역할을 하는 미들장비에서의 공정성을 다룬 DRFQ방식의 경우, 소수의 서버에 대해 공정성을 보장하기 위하여 제안된 방식으로서 보다 높은 상대적 서비스 요구사항을 갖는 서버만을 대상으로 공정성을 보장하고자 하며 이를 위하여 특정 요청이 더 많은 시간을 요청하는 자원의 서비스 시간을 통해 가상시간을 관리한다. 그러나 플래시 메모리의 요청은 버스와 칩을 동시에 사용하는 개시, 종료 구간과 칩만을 사용하는 구간으로 나누어져 있어 항상 특정 칩에서의 처리 시간이 버스에서의 처리시간 보다 길기 때문에 해당 기법의 제대로 된 구현이 불가능하다. 이와 같은 현상은 DRFQ 방식의 경우 모든 스트림이 소수의 자원에 대해 경쟁을 발생시키는 반면 플래시 메모리 시스템의 경우 다수의 자원이 존재하며 특정 스트림이 경쟁에 참여하지 않는 자원이 존재할 수 있는 동시에 각 자원별로 경쟁하는 스트림에 불균형이 존재할 수 있기 때문이다. 따라서 DRFQ방식은 플래시 메모리 시스템에 적용하는 데에 무리가 있다.

따라서 해당 환경에서 공정성 보장관리 단위와 서비스 가치의 상대성의 2가지 차원에서 바라본 4가지 관점의 이상적인 공정성 보장 모델을 제안하였다. 또한 해당 모델을 2차원 평면으로 해석하여 각 모델의 특성을 일정수준으로 반영하도록 하는 공정성 보장 분류 공간을

제안함으로써 사용자가 원하는 공정성 보장 정책을 선정할 수 있도록 자유도를 제공하였다.

또한 해당 이상적인 모델을 근사하여 동작하는 공정성 보장 기법을 제안하고 이를 높은 병렬성을 제공하는 무순서 플래시 메모리 제어기를 활용하는 플래시 공정성 보장 계층을 통해 플래시 메모리 시스템에 적용하였다. 이를 통해 플래시 메모리 저장장치에 대해 호스트의 요청과 내부관리 요청 사이의 경쟁을 공정하게 분배하는 플랫폼을 제안하였다.

해당 방식을 고해상 실시간 시뮬레이터 환경을 기반으로 구현한 결과 서비스 시간의 관점에서 공정한 처리를 보장함을 확인할 수 있었다. 또한 새로운 스트림이 경쟁에 참여할 때에 제안한 시스템에서의 응답시간이 기존 기법에 비해 줄어드는 것을 확인함으로써 내부동작 관리와 호스트 요청의 경쟁이 발생할 경우 호스트 요청의 응답시간을 보장할 수 있는 동작 특성을 확인하였다.

해당 연구는 공정성을 보장하는 하나의 플랫폼을 제안하였으며, 해당 플랫폼이 플래시 시스템에서 효율적으로 활용되기 위해서는 다음과 같은 후속 연구가 필요하다.

먼저 호스트 요청과 내부동작 사이에 부여되는 가중치를 관리하는 정책에 대한 연구가 필요하다. 호스트 요청의 응답시간 및 처리율을 보장함과 동시에 시스템의 동작을 유지하는데 필수적인 내부동작의 안정적인 진행을 위해서는 특정 시점에 호스트 요청과 내부유지 동작에 대한 가중치가 시스템의 상황에 따라 역동적으로 변화되어야 한다.

또한 FTL내부에서 유발되는 다양한 내부관리 요청을 추가적인 스트림으로 나누어 관리하는 정책에 대한 연구가 필요하다. FTL은 호스트 요청 이외에 사상정보와 같이 동작의 수행을 위하여 필수적인 속

성정보와 체크포인트와 같이 전원오류 등 문제 상황에 대처하기 위하여 관리하는 속성정보와 관련된 연산들을 생성한다. 또한 그리고 가용 물리 영역을 확보하기 위한 쓰레기 수집기능과 관련된 연산들, 비트반전 오류 등에 대처하기 위한 연산들 등 다양한 내부 연산으로 인해 플래시 메모리 연산이 생성된다. 이와 같은 내부관리 요청 사이의 경쟁도 해당 시점의 시스템의 요청 우선순위에 따른 관리가 필요하다.

더 나아가 보다 정확하고 효율적인 공정성 관리를 위하여 각 계층의 기능을 재분배 혹은 확장하는 연구가 필요하다. 무순서 플래시 메모리 제어기를 활용하지 않고 버스와 플래시 메모리 칩의 동작 제어만을 담당하는 하드웨어 제어기를 활용할 경우 칩의 스케줄링 정책과 공정성 보장 정책 모두를 관리하는 소프트웨어 제어 계층의 구현이 가능해진다. 이 경우 보다 높은 공정성과 효율적인 관리가 가능하리라 생각된다. 또한 FTL이 목표서버를 자유롭게 지정할 수 있는 쓰기 연산에 대한 부하 균등화를 공정성 보장에 대한 정보를 활용하여 제공함으로써 시스템의 효율을 증가시킬 수 있다.

참고 문헌

[1] J. Choi, and S. S. Kwang. 3D approaches for non-volatile memory. In *Proc. IEEE Symposium on VLSI Technology-Digest of Technical Papers*, 2011.

[2] D. A. Menasce. QoS issues in web services. *IEEE Internet Computing*, 6(6):72-75, 2002.

[3] J. U. Kang, J. S. Hyun, H. J. Maeng, and S. Y. Cho. The multi-streamed solid-state drive. In *Proc. 6th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)*, 2014.

[4] M. S. Jung, W. I. Choi, S. Srikantaiah, J. H. woo, and M. T. Kandemir. HIOS: a host interface I/O scheduler for solid state disks. In *Proc. 41st annual international symposium on Computer architecture (ISCA)*, 2014.

[5] E. H. Nam, S. J. Kim, H. S. Eom, and S. L. Min. Ozone (o3): An out-of-order flash memory controller architecture. *IEEE Transactions on Computers*, 60(5):653-666, 2011.

[6] 배종보. 플래시 저장장치의 성능 및 신뢰성 평가를 위한 무순서 플래시 메모리 제어기의 고해상 실시간 시뮬레이션. *서울대학교 공학 석사학위논문*, 2015.

[7] J. S. Kim, J. M. Kim, S. H. Noh, S. L. Min, and Y. K. Cho. A space-efficient flash translation layer for CompactFlash systems. *IEEE Transactions on Consumer Electronics*, 48(2):366-375, 2002

[8] C. I. Park, P. Talawar, D. S. Won, M. J. Jung, J. B. Im, S. S. Kim, and Y. J. Choi. A high performance controller for NAND flash-based solid state disk (NSSD). In *Proc. IEEE Non-Volatile Semiconductor Memory Workshop*, 2006.

[9] K. S. Ha, J. Y. Jeong, and J. H Kim. A read-disturb management technique for high-density NAND flash memory. In *Proc. 4th Asia-Pacific Workshop on Systems (APSYS)*, 2013.

[10] L. M. Grupp, A. M. Caulfield, J. Coburn, and S. Swanson. Characterizing flash memory: anomalies, observations, and applications. In *Proc. 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2009.

[11] A. Gupta, Y. J. Kim, and B. Urgaonkar. DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings. *ACM SIGPLAN notices,* 44(3):229-240, 2009.

[12] M. Chiang, C. H. Lee, and R. Chang. Using data clustering to improve cleaning performance for flash memory. *Software: Practice & Experience,* 29(3):267-290. 1999.

[13] H. Zhang. Service disciplines for guaranteed performance service in packet-switching networks. *Proceedings of the IEEE*, 83(10):1374-1399, 1995.

[14] J. Nagle. On packet switches with infinite storage. *IEEE Transactions on Communications*, 35(4):435-438, 1987.

[15] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. *ACM SIGCOMM*

Computer Communication Review, 19(4):1-12, 1989.

[16] A. K. Parekh, and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Transactions on Networking*, 1(3):344-357, 1993.

[17] S. J. Golestani. A self-clocked fair queueing scheme for broadband applications. In *Proc. IEEE INFOCOM'94 Conference on Computer Communications,* 1994.

[18] P. Goyal, H. M. Vin, and H. Cheng. Start-time fair queueing: a scheduling algorithm for integrated services packet switching networks. *ACM SIGCOMM Computer Communication Review*, 26(4):157-168, 1996.

[19] J. C. R. Bennett, and H Zhang. WF^2Q: worst-case fair weighted fair queueing. In *Proc. IEEE INFOCOM'96 Conference on Computer Communications*, 1996.

[20] N. Ni, and L. N. Bhuyan. Fair scheduling in internet routers. *IEEE Transactions on Computers,* 51(6):686-701, 2002.

[21] J. M. Blanquer, and B. Özden. Fair queuing for aggregated multiple links. *ACM SIGCOMM Computer Communication Review*, 31(4):189-197, 2001.

[22] A. Ghodsi, V. Sekar, M. Zaharia and I. Stoica. Multi-resource fair queueing for packet processing. *ACM SIGCOMM Computer Communication Review*, 42(4):1-12, 2012.

[23] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant Resource Fairness: Fair

Allocation of Multiple Resource Types. In *Proc. 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2011.

[24] C. A. Waldspurger, and W. E. Weihl. Stride scheduling: Deterministic proportional share resource management. *Technical report*, MIT Laboratory for Computer Science, 1995.

[25] C. S. Pabla. Completely fair scheduler. *Linux Journal*, 2009(184):4, 2009.

[26] 이은규, 원선, 김강희, 남이현. 플래시 저장 장치의 QoS 지원을 위한 플래시 연산 그룹(Flash Operation Groups) 스케줄링. *정보과학회*, 2015.

Abstract

Fair Queueing Flash Memory Control Scheme Providing Stable Response Times of Host Requests

Ansu Na
Department of Electrical Engineering and Computer Sciences

The Graduate School
Seoul National University

Performance of flash memory storage system, such as response time and throughput, can be degraded because of the competition between flash memory operations issued for host interface requests and those issued for flash memory system internal-managements.

Such performance degradations can be critical to systems such as cloud servers and data centers, so a systematic solution to the problem must be provided.

This thesis proposes the fair queuing flash memory control scheme for providing stable response time and throughput. To implement a fair queuing algorithm, an ideal fairness model must be specified. So the ideal fairness model space for the destination assigned multi-server environment is proposed. The fairness model space has two dimensions: the fairness providing unit and the relativeness of service value.

System designers can choose the fairness model according to their own need. And the fairness management module is implemented based on the ideal fairness model, which provides fairness between request streams of the flash memory storage system.

By implementing the proposed module on a high-fidelity and real-time flash memory simulator, it was shown that the service time between requests streams can be shared fairly by using the fairness management module. In addition, a dramatic decrease in response time compared to existing schemes were shown through experiments measuring the response time of new streams competing for the flash memory storage system.

Keywords : Flash Memory Storage, Fair Queueing Algorithm, QoS(Quality of service), FTL(Flash Translation Layer), Flash Memory Controller

Student number: 2013-23113